Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Розгортання токену SuperChainErc20
за допомогою OP Stack

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-3
Микола ДІМІТРОВ
(власне ім'я, прізвище)

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

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

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

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

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

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

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

Освітня програма Комп’ютерна інженерія
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дімітрову Миколі Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розгортання токену SuperChainErc20 за допомогою OP Stack

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

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

3. Вхідні дані до роботи 1) ERC-20 документація; 2) OP Stack docs; 3) OpenZeppelin; 4) Foundry; 5) RPC від Alchemy

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз сучасних фреймворків для створення смарт-контрактів;
2) реалізація та тестування токену за стандартом ERC-20;
3) проведення деплойменту в мережі Optimism Sepolia та Base Sepolia;
4) перевірка функціональності, безпеки та доступності токену через Web3-гаманці;
5) аналіз результатів багатомережевого тестування.

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

Слайд-презентація – 19

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Формулювання мети та завдань дослідження	26.05.25–27.05.25	
2	Вибір інструментів та налаштування Foundry	28.05.25–29.05.25	
3	Розробка смарт-контракту SuperChainERC20	30.05.25–01.06.25	
4	Тестування функцій mint, burn, transfer	02.06.25–03.06.25	
5	Розгортання контракту в мережах OP Stack	04.06.25–05.06.25	
6	Верифікація та інтеграція з MetaMask	06.06.25–07.06.25	
7	Оформлення пояснювальної записки	08.06.25–10.06.25	
8	Подання роботи на перевірку та рецензію	18.06.25–20.06.25	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 58 с., 10 рис., 1 дод., 21 джерел.

ETHEREUM, СМАРТ-КОНТРАКТ, ERC-20, ТОКЕН, FOUNDRY, OP STACK, OPTIMISM, BASE, SOLIDITY.

Метою кваліфікаційної роботи є створення, розгортання та тестування смарт-контракту стандарту ERC-20 із додатковими функціями керування, з подальшим його впровадженням у масштабовані блокчейн-мережі другого рівня, зокрема Optimism та Base, побудовані на технології OP Stack.

У ході виконання кваліфікаційної роботи буде розроблено, протестовано та розгорнуто смарт-контракт стандарту ERC-20 під назвою SuperChainERC20 з підтримкою функцій емісії та спалювання токенів. Проєкт реалізується з використанням фреймворку Foundry у тестових мережах OP Stack (Optimism Sepolia та Base Sepolia), що дозволяє перевірити працездатність рішення в умовах масштабованих L2-мереж. Особливу увагу приділено архітектурі проєкту, безпечності реалізації та автоматизації процесів деплойменту та верифікації.

ABSTRACT

Bachelor’s thesis: 58 pages, 10 figures, 1 appendices, 21 sources.

ETHEREUM, SMART CONTRACT, ERC-20, TOKEN, FOUNDRY, OP STACK, OPTIMISM, BASE, SOLIDITY.

The major goal of this thesis is to design, implement, and deploy an ERC-20 standard smart contract named SuperChainERC20 with extended functionality, including token minting and burning mechanisms. The solution is aimed at demonstrating compatibility with OP Stack infrastructures such as Optimism Sepolia and Base Sepolia.

In order to achieve this, a full development workflow using the Foundry framework was followed, including project initialization, environment configuration, contract compilation, unit testing in Solidity, deployment to L2 testnets, and contract verification. Special attention was given to security controls, gas efficiency, and readiness for integration into Web3 ecosystems.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ABI – інтерфейс двійкових додатків (англ. Application Binary Interface)

API – інтерфейс програмування додатків (англ. Application Programming Interface)

CLI – інтерфейс командного рядка (англ. Command Line Interface)

DA – доступність даних (англ. Data Availability)

DAO – децентралізована автономна організація (англ. Decentralized Autonomous Organization)

dApp – децентралізований додаток (англ. decentralized application)

DeFi – децентралізовані фінанси (англ. Decentralized Finance)

EIP – пропозиція покращення Ethereum (англ. Ethereum Improvement Proposal)

ERC – запит коментарів Ethereum (англ. Ethereum Request for Comments)

EVM – віртуальна машина Ethereum (англ. Ethereum Virtual Machine)

L1 – перший рівень (англ. Layer 1)

L2 – другий рівень (англ. Layer 2)

NFT – невзаємозамінний токен (англ. Non-Fungible Token)

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

UI – користувацький інтерфейс (англ. User Interface)

Web3 – веб третього покоління (англ. Web 3.0)

ВСТУП

Стрімкий розвиток децентралізованих технологій, зокрема блокчейн-систем, відкрив нові горизонти для побудови інфраструктури, яка здатна функціонувати без централізованих посередників. На тлі зростаючої популярності цифрових активів, смарт-контрактів та децентралізованих фінансів (DeFi), платформа Ethereum закріпила за собою роль флагманського середовища для розробки блокчейн-додатків нового покоління. Її гнучка архітектура, розширюваність та підтримка стандартизованих інтерфейсів (зокрема ERC-20) дозволили сформувати цілу екосистему, що активно розвивається у напрямі Web3.

Однак, поряд із зростанням навантаження на мережу Ethereum, постало питання її масштабованості, що вивело на передній план технології другого рівня (Layer 2). Одним із найперспективніших рішень у цьому напрямку стали оптимістичні ролапи (Optimistic Rollups) - механізм, що дозволяє зберігати сумісність з EVM та значно зменшити вартість транзакцій шляхом виведення обчислень за межі основного ланцюга. Саме на цій технології базується OP Stack - модульний open-source фреймворк для створення власних L2-мереж, що є фундаментом для масштабованої міжмережевої екосистеми Superchain.

У цьому контексті особливої уваги набуває процес створення та розгортання кастомних токенів, які повинні залишатися сумісними як із базовою логікою Ethereum, так і з новими середовищами виконання у L2-ланцюгах. Токен SuperChainERC20, реалізований у межах цієї практики, є прикладом прикладного використання OP Stack з урахуванням усіх викликів та можливостей, що надає багаторівнева архітектура сучасного блокчейну.

1 БЛОКЧЕЙН ETHEREUM

1.1 Генезис Ethereum

Ідея створення Ethereum виникла як відповідь на обмеження першого покоління блокчейн-систем, зокрема Bitcoin, який забезпечував лише обмежену функціональність передачі вартості між учасниками без можливості реалізувати умовну, гнучку логіку[1]. У 2013 році молодий програміст Віталік Бутерін опублікував маніфест, у якому запропонував концепцію «світового комп'ютера»[2] - універсального середовища, де будь-хто може розмістити свою програму у вигляді смарт-контракту, яка буде гарантовано виконана блокчейн-мережею у децентралізований спосіб.

Основною ідеєю Ethereum було створення не просто криптовалюти, а платформи загального призначення, яка б дозволяла реалізовувати довільну логіку (умовні інструкції, цикли, перевірки тощо) без необхідності створювати власний блокчейн для кожного окремого додатку. Це вирішувало одразу кілька проблем: обмеженість функціоналу інших криптовалют, відсутність модульності, складність розробки нових блокчейн-систем.

Ethereum був задуманий як Turing-повна платформа, здатна обробляти складні обчислювальні процеси через власну віртуальну машину (Ethereum Virtual Machine) та надавати спільноті розробників можливість створювати децентралізовані додатки (dApps) з вбудованими правилами, які виконуються автоматично.

На відміну від Bitcoin, де єдиним доступним видом транзакцій є передача токенів від одного користувача до іншого, Ethereum з самого початку був орієнтований на виконання програмного коду, розміщеного в мережі. Це дозволило створити нові бізнес-моделі: від децентралізованих бірж до автономних організацій (DAO), від NFT до протоколів кредитування, все це без втручання централізованих структур.

Запуск Ethereum у 2015 році був проривним не лише з технологічної точки зору[3], але й з філософської. Його творці поставили перед собою мету перерозподілити цифрову владу, зробити інтернет менш централізованим та підконтрольним великим корпораціям, створивши інфраструктуру, де будь-хто може без дозволу створити ціннісну логіку та бути впевненим у її дотриманні.

Таким чином, генезис Ethereum пов’язаний із народженням нової парадигми - дозвільного, універсального, самовиконуваного коду в децентралізованій інфраструктурі, який можна сприймати як фундамент для формування Web3 - інтернету нового покоління, де контроль над даними і логікою переходить до користувача.

1.2 Смарт-контракти в децентралізованих

Однією з ключових інновацій, що визначили унікальність та функціональну перевагу Ethereum над блокчейн-системами попереднього покоління, є впровадження смарт-контрактів як базової одиниці програмної логіки. Смарт-контракт це комп’ютерна програма, що зберігається у блокчейні та автоматично виконується при настанні певних умов, визначених у коді. У контексті Ethereum смарт-контракти стали фундаментом для побудови децентралізованих застосунків (dApps), а згодом і цілих екосистем, що функціонують без потреби в централізованому контролі.

Завдяки Ethereum Virtual Machine (EVM), яка забезпечує ізольоване й детерміноване середовище виконання, смарт-контракти можуть взаємодіяти між собою, викликати функції інших контрактів, зберігати дані у власному стані (storage), та реагувати на події в мережі. Це дозволяє реалізувати складну прикладну логіку: від простих токенів до складних протоколів кредитування або децентралізованого управління.

Роль смарт-контрактів у децентралізованій екосистемі можна охарактеризувати за кількома напрямками:

- Смарт-контракти дозволяють замінити посередників (біржі, платіжні системи, арбітражні інституції) алгоритмічною логікою, яка виконується об’єктивно й неупереджено;

- Код контракту доступний для перегляду, верифікації та аудиту, що підвищує рівень довіри з боку користувачів та інвесторів;

- За допомогою смарт-контрактів реалізуються цілі економічні моделі: емісія токенів, виплата винагород, управління DAO, токеноміка NFT, farming, lending тощо;

- Смарт-контракти можуть взаємодіяти між собою без участі користувача. Це відкриває шлях до формування композиційних протоколів (DeFi Legos), які об'єднуються у нові продукти й сервіси.

Поява стандартів, зокрема ERC-20 (для взаємозамінних токенів) та ERC-721/1155 (для NFT), заклала підґрунтя для інтероперабельності та повторного використання коду[5]. Завдяки цьому тисячі децентралізованих додатків можуть працювати на єдиній базі - Ethereum, не створюючи розривів у сумісності між собою.

У ширшому контексті, смарт-контракти виконують роль правових та організаційних фреймворків у цифровому просторі: вони можуть фіксувати угоди, управляти доступом до даних, нараховувати прибутки, або ж регулювати участь у виборах. Це забезпечує перехід від централізованих платформ до відкритих протоколів, де правила встановлюються не корпорацією, а кодом, що не підлягає зміні без згоди мережі. Принцип роботи зображен на рисунку 1.1

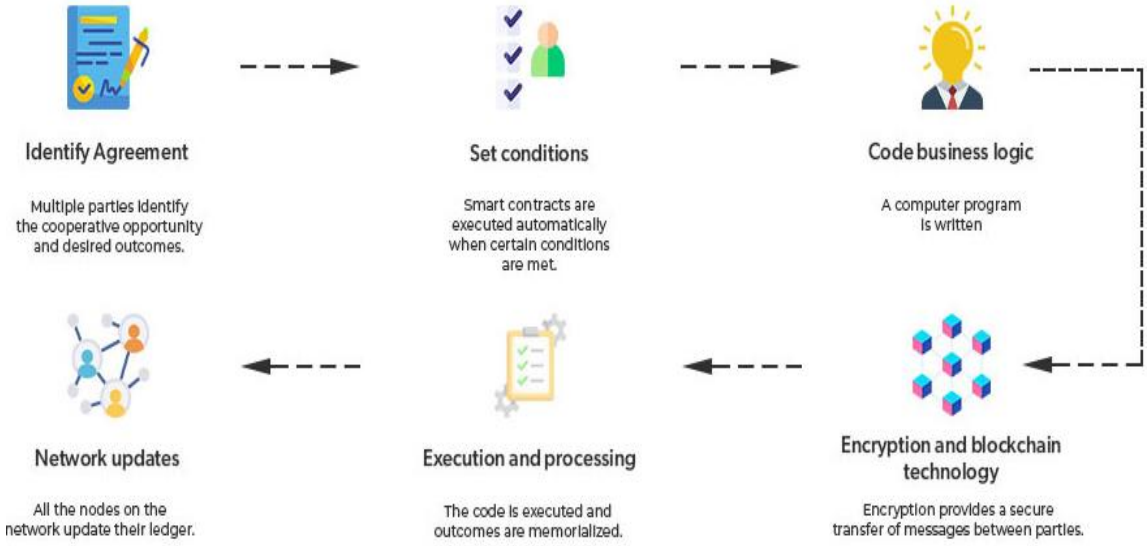

Рисунок 1.1 – Послідовність роботи смарт-контракту

Таким чином, смарт-контракти стали архітектурною основою Web3, забезпечуючи децентралізацію не лише на рівні зберігання й верифікації даних, а й на рівні логіки взаємодії, економічної координації та управління спільнотами.

1.3 Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) є ядром архітектури платформи Ethereum і виконує ключову функцію забезпечення обчислювального середовища для запуску смарт-контрактів. Вона слугує універсальним механізмом інтерпретації програмного коду, що гарантує однакове виконання логіки на кожному вузлі мережі, незалежно від операційної системи чи апаратного забезпечення. Саме завдяки EVM платформа Ethereum змогла реалізувати концепцію «світового комп’ютера», надаючи розробникам можливість створювати складні децентралізовані застосунки без потреби створювати окремі блокчейни.

EVM - це стекова віртуальна машина, яка працює з байт-кодом, згенерованим із мов програмування високого рівня (насамперед Solidity)[3]. Усі інструкції, змінні, операції над пам’яттю та передача повідомлень

відбуваються через стек - структуру даних типу LIFO (last-in, first-out). Віртуальна машина ізольована від основного коду ноди, що унеможливлює втручання у внутрішні процеси вузла і підвищує безпеку. Структура та логіка виконання Ethereum Virtual Machine зображена на рисунку 1.2

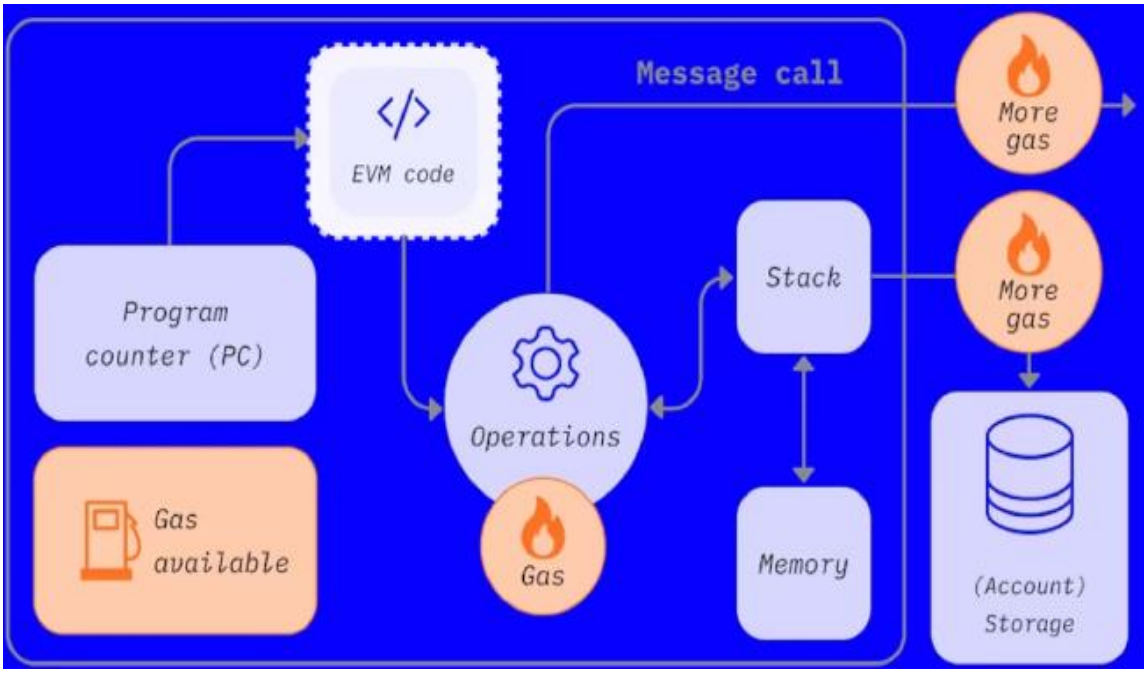

Рисунок 1.2 – Принцип роботи EVM

Одна з визначальних переваг EVM це детермінованість обчислень: результат виконання програми в будь-якому вузлі буде однаковим, якщо вхідні дані не змінюються. Це фундаментальна властивість для консенсусу у розподіленому середовищі.

До основних функцій EVM належать:

- Інтерпретація та виконання смарт-контрактів у верифікованому середовищі;

- Обробка транзакцій, що містять інструкції виклику функцій контрактів;

- Керування станом блокчейну, зокрема запис/читання змінних, створення нових контрактів, обробка подій (events);

- Обчислення вартості виконання за допомогою механізму gas, який

обмежує споживання ресурсів та запобігає нескінченним циклам.

Також важливою є роль EVM як універсального середовища для крос-мережевого розгортання контрактів. Завдяки тому, що численні інші блокчейни (BNB Chain, Avalanche C-Chain, Polygon, Fantom тощо) підтримують сумісність з EVM[7], розробники можуть легко переносити застосунки між мережами, використовуючи однаковий код та інструменти. Це сприяло формуванню поняття EVM-сумісної екосистеми - інфраструктурного простору, в якому домінує загальна логіка, стандарти і модулі.

У контексті практики з розгортання токена SuperChainERC20 варто зазначити, що підтримка EVM є критично важливою для функціонування рішень другого рівня, зокрема тих, які побудовані на базі OP Stack. Оскільки OP Stack реалізує віртуальну машину, повністю сумісну з EVM, це дозволяє без змін переносити існуючі контракти з основного ланцюга Ethereum у модульні L2-мережі, що входять до Superchain.

Таким чином, EVM виступає не лише технічним інтерпретатором коду, а й архітектурною основою сумісності, безпеки та стандартизації децентралізованих додатків у ширшому масштабі блокчейн-інфраструктур.

1.4 Етапи розвитку мережі Ethereum

Ethereum як децентралізована платформа зазнала значної еволюції від моменту свого запуску у 2015 році. Кожен етап її розвитку супроводжувався важливими технічними оновленнями, які поступово підвищували масштабованість, безпеку, ефективність і загальну стабільність мережі. У цьому підпункті розглянуто основні етапи становлення Ethereum: від експериментального релізу Frontier до останніх масштабних оновлень Cancun–Deneb та Petra.

1.4.1 Frontier

Перший публічний запуск Ethereum позначив початок функціонування платформи як блокчейну з підтримкою смарт-контрактів. На цьому етапі були реалізовані базові механізми: майнінг, транзакції, створення контрактів. Хоча функціональність залишалась обмеженою, Frontier став важливим середовищем для тестування EVM і для перших dApp-розробників.

1.4.2 Homestead

Homestead став першою «стабільною» версією Ethereum. Оновлення принесло удосконалення протоколу, зокрема безпечніший механізм створення контрактів і поліпшене управління мережею. Це сприяло зростанню довіри з боку розробників і стало відправною точкою для комерційного використання платформи.

1.4.3 Metropolis

У межах оновлення Metropolis було реалізовано дві великі фази:

- Byzantium додало підтримку zk-SNARKs, нові інструкції в EVM та зниження комісій;
- Constantinople оптимізувало зберігання даних у контрактах, знизило винагороду за блок та підготувало мережу до переходу на Proof of Stake.

Ці зміни перетворили Ethereum з експериментальної платформи на стабільну інфраструктуру для DeFi і токен-економіки.

1.4.4 Istanbul, Berlin, London

Ці три оновлення включали низку EIP-покращень:

- Підвищення сумісності з масштабувальними рішеннями (зокрема zkRollups);

- Оптимізація газової системи;
- Впровадження EIP-1559, що змінило модель комісій: частина gas тепер спалюється, роблячи ефір дефляційним активом.

Ці оновлення підготували мережу до повного переходу на Proof of Stake.

1.4.5 The Merge

The Merge став визначальним моментом в історії Ethereum. Вперше у світовій практиці публічний блокчейн без хардфорку перейшов з Proof of Work на Proof of Stake. Основні наслідки:

- Споживання енергії знизилось на ~99,95%;
- Запроваджено механізм стейкінгу для валідаторів;
- Мережа залишилась повністю сумісною з EVM і наявними контрактами.

Це оновлення закріпило Ethereum як екологічно безпечну та стійку платформу для довгострокового масштабування.

1.4.6 Cancun–Deneb

Оновлення Cancun реалізувало EIP-4844 (Proto-Danksharding), який дозволив оптимістичним і zk-ролапам записувати великі масиви даних у вигляді blob-транзакцій, не навантажуючи базовий ланцюг[7]. Ефекти:

- Зниження вартості транзакцій у L2;
- Підготовка до повного шардінгу даних;
- Зміцнення позиції Ethereum як L1 для модульних L2-систем.

1.4.7 Petra

Petra найновіше велике оновлення, яке значно розширило архітектуру

Ethereum як data-availability шару[4]. Було додано:

- Розширення Danksharding, повна підтримка blob storage;
- Нові EVM-опкоди та системи доказів для rollup-мереж (включно з zkEVM);
- Сумісність із Superchain та модульними фреймворками (OP Stack, Polygon CDK, zkStack);

Petra закріпила модель, де більшість обчислень переноситься в Layer 2, а основна мережа гарантує безпеку та доступність даних.

2 ТЕХНОЛОГІЯ OP STACK

2.1 OP Stack як фундамент інфраструктури Superchain

З розвитком екосистеми Ethereum та актуалізацією потреби в масштабованості, значну увагу привернула концепція модульної архітектури блокчейн-систем, де обчислення, консенсус і збереження даних розділяються між різними рівнями. В цьому контексті одним із найважливіших технологічних рішень останніх років став OP Stack - відкритий модульний фреймворк для створення оптимістичних ролап-мереж, який розроблено командою Optimism Foundation[4]. Його місія полягає не лише у створенні окремої L2-мережі, а у побудові сумісної багатоланцюгової інфраструктури, яку самі розробники називають Superchain.

OP Stack - це програмна структура, яка забезпечує уніфіковану модель побудови блокчейн-ланцюгів другого рівня (L2), орієнтованих на використання Ethereum як базового шару безпеки (L1). Завдяки модульному дизайну, OP Stack дозволяє розробникам запускати власні L2-мережі, що залишаються повністю сумісними з Ethereum Virtual Machine (EVM) і при цьому користуються перевагами масштабування через ролапи. Архітектура OP Stack зображена на рисунку 2.1[4]

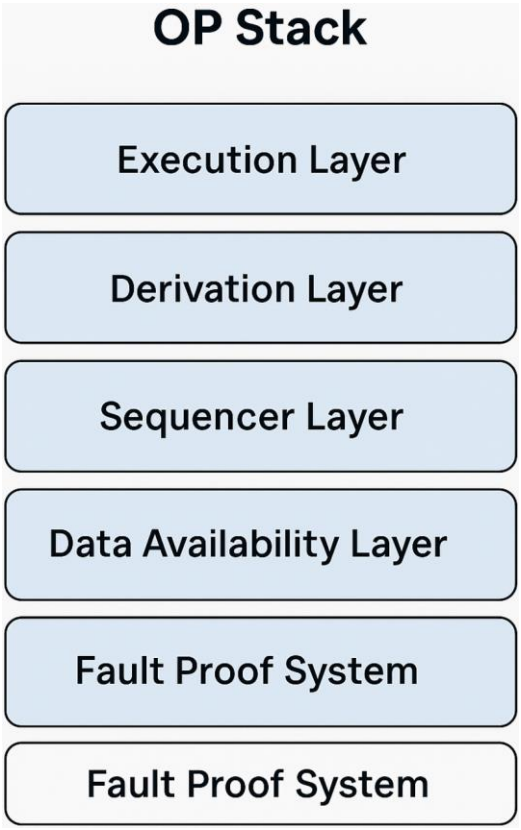

Рисунок 2.1 – Архітектура OP Stack та її рівнева структура

2.1.1 Основні складові OP Stack

Архітектура OP Stack поділяється на кілька чітко визначених рівнів:

- Execution Layer відповідає за обробку транзакцій та виконання смарт-контрактів. Він повністю сумісний з EVM, що дозволяє легко портовати будь-який Ethereum-застосунок на нову L2-мережу;

- Derivation Layer шар обробки даних із L1. Він агрегує та передає транзакції від користувачів, отримує дані з базової мережі Ethereum і формує пакет для L2;

- Sequencer Layer логіка упорядкування транзакцій. Він визначає послідовність виконання, забезпечуючи предикативність і низьку затримку;

- Data Availability Layer відповідальний за публікацію та збереження даних, необхідних для верифікації. У майбутньому може бути замінений на зовнішні DA-рішення (наприклад, EigenDA, Celestia);

- Fault Proof System (майбутній модуль) забезпечує механізм

оскарження (challenge) в оптимістичній моделі, коли транзакції можна перевірити після включення в ланцюг.

2.1.2 Концепція Superchain

Superchain це стратегічна ідея об’єднання всіх OP Stack-мереж у єдину координаційну інфраструктуру, де спільна мова це EVM і OP Stack. Учасники можуть запускати власні ролапи (наприклад, Base від Coinbase, Mode, Zora), але при цьому:

- Використовують єдиний стек технологій;
- Мають спільні інструменти та api;
- Підтримують централізовану децентралізацію через дозволену модель;

Superchain обіцяє масштабування Ethereum не через один величезний блокчейн, а через тисячі скоординованих малих, які взаємодіють між собою за стандартами OP Stack. Переваги використання OP Stack:

- Повна EVM-сумісність;
- Можливість замінювати окремі шари (наприклад, DA Layer);
- Зменшення вартості транзакцій через rollups;
- Швидкість запуску, готові шаблони, CLI, tooling;
- Розробка ведеться у відкритому GitHub-репозиторії, рішення перевірені battle-tested застосунками.

2.2 Стандартизація токенів у мережі Ethereum

Однією з ключових передумов широкого поширення блокчейн-застосунків, зокрема у сфері децентралізованих фінансів (DeFi), є уніфікація форматів взаємодії між контрактами. У цьому контексті стандарти токенів Ethereum, насамперед ERC-20, відіграють фундаментальну роль, слугуючи основою для випуску цифрових активів, інтеграції з біржами, гаманцями та

смарт-протоколами.

2.2.1 ERC-20 як основа токенізації

Стандарт ERC-20 був запропонований у 2015 році як інтерфейс взаємодії для взаємозамінних токенів (fungible tokens) [5]. Його основна мета - створити єдиний набір функцій, які мають бути реалізовані кожним токеном: balanceOf, transfer, approve, transferFrom, allowance. Цей підхід забезпечив:

- Сумісність з будь-якими defi-протоколами та гаманцями, що підтримують стандарт;
- Стабільну логіку взаємодії для бірж, DAO, NFT-платформ;
- Масштабування ринку завдяки єдності коду та поведінки токенів.

Власне, завдяки ERC-20 стало можливим швидке створення тисяч токенів (USDT, DAI, UNI, LINK тощо) без необхідності розробляти окрему мережу чи інфраструктуру.

2.2.2 Вимоги до контрактів і сумісність з L2

З розвитком мереж другого рівня (L2), зокрема на основі OP Stack, виникла необхідність у забезпеченні повної сумісності токенів ERC-20 з мультишаровими системами. Це означає, що токен повинен:

- Мати однакову логіку як у L1, так і у L2 середовищі;
- Підтримувати бріджинг (cross-chain transfers) з використанням bridge-контрактів;
- Правильно обробляти контексти виклику (msg.sender, msg.value) у rollup-мережах;
- Взаємодіяти з L2-оптимізованими інструментами наприклад, optimismportal, l2standardbridge.

У межах практики зі створення SuperChainERC20, було реалізовано

токен на основі OpenZeppelin-бібліотеки з повною підтримкою ERC-20, а також протестовано сумісність із Optimism-базованим стеком через L2 Gateway[6].

2.2.3 Безпека та аудит смарт-контрактів

Зважаючи на критичну роль токенів у фінансових операціях, безпека контракту є обов'язковою умовою для його запуску в публічному середовищі. Основні практики:

- Аудит коду сторонніми командами (Certik, openzeppelin Audit, Trail of Bits)[8];
- Використання battle-tested бібліотек, як-от openzeppelin Contracts;
- Оголошення функцій як external замість public там, де це можливо;
- Встановлення обмежень на кількість емісії, доступ до mint/burn функцій, паузу контракту тощо.

У контексті Superchain, де L2-мережі функціонують як розширення базового Ethereum, уніфікація токенів та перевіреність їх логіки є необхідною передумовою довіри до будь-якого нового активу, що з'являється у системі.

Таким чином, стандартизація токенів у мережі Ethereum, в першу чергу через ERC-20, забезпечила масове прийняття токенів у блокчейн-інфраструктурі. У свою чергу, адаптація цих стандартів до специфіки L2-технологій (як-от OP Stack) гарантує збереження функціональності, безпеки та взаємодії в мультишаровому середовищі, створюючи фундамент для масштабованої децентралізованої економіки.

2.3 Інтеграція інтелектуальних вимірювальних систем з блокчейн-технологією

Зі зростанням екологічних загроз та антропогенного навантаження на довкілля виникає потреба у створенні високоточних, масштабованих та

автономних рішень для моніторингу навколишнього середовища. Інтелектуальні вимірювальні системи (ІВС) забезпечують безперервний збір та аналіз екологічних даних у режимі реального часу. Їхній успіх значною мірою зумовлений інтеграцією сенсорних мереж, технологій IoT, машинного навчання та хмарних обчислень.

У той же час, забезпечення цілісності, прозорості та незмінності екологічних даних вимагає впровадження технологій Web3, зокрема блокчейн-платформ. Одним з ефективних рішень для цього є використання стеку OP Stack - модульної архітектури, що лежить в основі мереж Optimism та Base, які побудовані як L2-рішення над Ethereum.

У межах даного дослідження було реалізовано смарт-контракт токену SuperChainERC20 на базі стандарту ERC-20, з подальшим його розгортанням у мережах Optimism Sepolia та Base Sepolia. Такий токен може виконувати роль одиниці обліку або винагороди в системах моніторингу довкілля: наприклад, за кожен переданий екологічний показник користувач або пристрій може отримувати токен як стимул за участь у децентралізованому зборі даних.

Інтеграція ІВС з блокчейн-мережами на базі OP Stack відкриває нові можливості:

- Прозора фіксація даних з екосенсорів з можливістю перевірки кожного запису;

- Використання смарт-контрактів для автоматизації винагород, обмеження доступу або запуску сповіщень у разі критичних значень;

- Токенізація участі в екологічному моніторингу та створення стимулюючої економіки;

- Прогнозування та динамічне управління ресурсами на основі об’єднаних даних.

Таким чином, поєднання інтелектуальних вимірювальних систем з Web3-технологіями, зокрема за допомогою розгортання власного токену на OP Stack, формує новий клас екологічно-орієнтованих блокчейн-рішень, де

прозорість, автономність та безпека виступають ключовими факторами ефективного моніторингу довкілля.[21]

3 ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ

3.1 Вибір інструментів для створення смарт-контрактів

3.1.1 Аналіз доступних фреймворків для розробки

Розробка смарт-контрактів в екосистемі Ethereum передбачає використання спеціалізованих фреймворків, які спрощують компіляцію, тестування та деплоймент коду на блокчейн. На сьогодні найбільш поширеними інструментами у цій сфері є Truffle, Hardhat та Foundry. Кожен із них має свої особливості, переваги та обмеження, які слід враховувати під час вибору технологічного стеку для конкретного проєкту.[11]

Truffle був одним із перших фреймворків для розробки Ethereum-контрактів і певний час залишався домінантним інструментом. Його особливістю є інтегрована система міграцій, шаблони для тестування на JavaScript та підтримка власної консольної оболонки. Однак на сьогодні Truffle втрачає актуальність через повільність, складну інтеграцію з сучасними CI/CD-пайплайнами та обмежену підтримку новітніх рішень на кшталт OP Stack.[12]

Hardhat став логічним продовженням еволюції фреймворків і набув значної популярності серед розробників. Він забезпечує гнучке середовище для розробки, підтримує плагіни (зокрема hardhat-deploy, hardhat-ethers) і має зручну інтеграцію з TypeScript. Завдяки модульності та активному розвитку, Hardhat часто використовується в сучасних DeFi-проєктах. Проте його виконання залежить від Node.js, а тести здебільшого пишуться мовами загального призначення (JS/TS), що ускладнює написання специфічних сценаріїв для Solidity.

Foundry - найновіший з розглянутих фреймворків, створений мовою Rust. Він вирізняється надзвичайно високою продуктивністю та орієнтацією

на саму мову Solidity. Усі тести, скрипти та взаємодія з контрактами можуть бути реалізовані безпосередньо на Solidity, що спрощує підтримку коду і полегшує аудит. Foundry має вбудовану підтримку основних EVM-сумісних мереж, включаючи ті, що базуються на OP Stack (наприклад, Optimism або Base), а також інструменти для статичного аналізу та fuzz-тестування.[13]

Таким чином, на етапі попереднього аналізу було розглянуто три основні інструменти - Truffle, Hardhat і Foundry. Подальший вибір фреймворку здійснювався з урахуванням вимог до швидкості, безпеки, підтримки Solidity-орієнтованого тестування та сумісності з інфраструктурою OP Stack, що стало предметом наступного підпункту.

3.1.2 Переваги та доцільність вибору Foundry

Після проведення порівняльного аналізу фреймворків для розробки смарт-контрактів, було обрано Foundry як основний інструмент реалізації токену SuperChainERC20. Це рішення ґрунтується на низці технічних, продуктивних та організаційних переваг, які роблять Foundry найбільш придатним для сучасного стеку, орієнтованого на OP Stack.

Однією з ключових переваг Foundry є висока продуктивність. Оскільки фреймворк написаний мовою програмування Rust, він працює набагато швидше за інші інструменти, особливо при компіляції, тестуванні та виконанні великих наборів контрактів. Це дозволяє розробнику оперативно ітеративно вносити зміни до коду та миттєво перевіряти результати.[14]

Особливу увагу варто приділити підтримці тестування безпосередньо на мові Solidity. У порівнянні з іншими інструментами, де тести здебільшого пишуться на JavaScript або TypeScript, Foundry дозволяє писати тести в самих контрактах, використовуючи бібліотеку forge-std. Це полегшує підтримку тестів, підвищує точність перевірки логіки контрактів та забезпечує більшу прозорість для зовнішнього аудиту.

Foundry також пропонує структуровану архітектуру проєкту, що

включає директорії src/ (основний код контрактів), script/ (деплоймент-скрипти), test/ (тестові контракти) та конфігураційні файли (foundry.toml). Це сприяє зрозумілій організації проєкту, спрощує навігацію в репозиторії та забезпечує повторюваність експериментів у командній роботі.

Крім того, фреймворк включає широкий набір CLI-інструментів (forge, cast, anvil), що дозволяє взаємодіяти з блокчейном, запускати локальні мережі, підписувати транзакції, перевіряти ABI та працювати з on-chain-даними без потреби в зовнішніх бібліотеках. Зокрема, інструмент forge підтримує автоматичну генерацію ABI, аналіз коду (forge inspect), перевірку покриття (forge coverage) та fuzz-тестування.[15]

Не менш важливою перевагою є повна сумісність з OP Stack, включаючи мережі Optimism та Base, без необхідності використання додаткових плагінів. Foundry безпосередньо підтримує роботу з будь-яким RPC-сумісним середовищем, включаючи тестові мережі, такі як Optimism Sepolia та Base Sepolia. Це спрощує деплоймент у L2-мережі та прискорює розробку.

Таким чином, вибір Foundry у межах даної роботи є технічно обґрунтованим, забезпечує продуктивну розробку, високу надійність та повну сумісність із цільовою інфраструктурою OP Stack.

3.2 Архітектура рішення та структура проєкту

Ефективна організація архітектури проєкту є необхідною умовою для розробки, масштабування та підтримки смарт-контрактної системи. У межах реалізації токену SuperChainERC20 архітектура рішення була побудована з урахуванням принципів модульності, повторюваності та сумісності з інфраструктурою OP Stack. Для реалізації було обрано фреймворк Foundry, який пропонує структуровану файлову організацію та чіткий розподіл функціональних компонентів.

Проєкт має класичну структуру, прийняту в екосистемі Foundry:

- src основна директорія, що містить смарт-контракти. У ній знаходиться файл SuperChainERC20.sol, який реалізує логіку стандарту ERC-20 з можливістю налаштування імені токена, символу, початкової емісії та прав доступу (owner);

- test містить тести, написані на мові Solidity з використанням бібліотеки forge-std. У тестах перевіряється коректність початкового балансу, переказів, дозволів (approve, transferFrom) та додаткових функцій (наприклад, mint або burn, якщо вони реалізовані);

- script включає скрипти для розгортання контракту на вибрані блокчейн-мережі. Наприклад, Deploy.s.sol містить логіку ініціалізації контракту, підключення до RPC та підпису транзакції з використанням приватного ключа;

- foundry.toml конфігураційний файл проєкту, де задаються версії Solidity, шляхи до джерел коду, адреси RPC-ендпоінтів та інші глобальні параметри;

- env файл із приватними змінними середовища, що містить приватні ключі, API-токени, адреси RPC і використовується у скриптах через бібліотеку dotenv.

Архітектура рішення побудована з урахуванням багатомережевої підтримки. Розгортання токена передбачено як у тестовій мережі Optimism Sepolia, так і в Base Sepolia, що дозволяє порівнювати параметри роботи, вартість газу та швидкість підтвердження транзакцій у різних L2-мережах.

Весь проєкт реалізується відповідно до принципів open-source: структура коду зрозуміла та легко адаптується під майбутні розширення (наприклад, інтеграція в децентралізовані біржі або додавання NFT-функціоналу поверх токену).[16]

Таким чином, архітектура рішення дозволяє реалізувати повний цикл створення, тестування та розгортання токену, при цьому забезпечуючи високу масштабованість і підтримку сучасних технологічних стандартів.

3.3 Обґрунтування вибору цільових мереж

3.3.1 OP Stack як технологічна база

На момент реалізації даної кваліфікаційної роботи екосистема Ethereum активно розвивається у напрямку масштабування за допомогою рішень другого рівня (Layer 2). Одним із найбільш перспективних і технологічно стабільних підходів у цьому контексті є OP Stack - відкритий модульний стек компонентів, що лежить в основі мережі Optimism та її похідних (Base, Zora, Mode тощо).

OP Stack було розроблено з метою створення уніфікованого фреймворку для побудови масштабованих блокчейн-мереж, сумісних з Ethereum. Його головними перевагами є:

- Контракти, розгорнуті на OP Stack, не потребують модифікацій порівняно з Ethereum Mainnet;
- Дозволяє адаптувати окремі компоненти під специфіку проєкту, включаючи sequencer, data availability layer тощо;
- Висока пропускна здатність і низька вартість транзакцій за рахунок ролап-архітектури та агрегування даних.

Окрім технічних переваг, OP Stack має широку підтримку з боку спільноти та провідних компаній, включаючи Coinbase (мережа Base), а також активну документацію, інструменти моніторингу та доступ до тестових мереж. Така інфраструктурна зрілість робить OP Stack логічною відправною точкою для реалізації токенів, децентралізованих додатків та інших рішень на базі Ethereum.[17]

У межах цієї роботи було вирішено реалізовувати токен SuperChainERC20 безпосередньо у L2-середовищах, створених на базі OP Stack, що дозволяє поєднати переваги ефективності з прозорістю та безпечністю головної мережі Ethereum.

3.3.2 Optimism Sepolia та Base Sepolia як тестових середовищ

Для перевірки працездатності токену SuperChainERC20 було обрано дві цільові тестові мережі: Optimism Sepolia та Base Sepolia. Обидві мережі є реалізаціями OP Stack та забезпечують повноцінне середовище для експериментів у масштабованому L2-контексті з низькими комісіями та швидким фіналізацією транзакцій.[18]

Optimism Sepolia - це офіційний тестнет Optimism, побудований поверх Ethereum Sepolia. Він є актуальним і підтримується розробниками як рекомендоване середовище для перевірки контрактів до розгортання в основній мережі Optimism. Серед ключових переваг Optimism Sepolia:

- Повна сумісність із мережевим стеком Ethereum (EVM, RPC, ABI);
- Наявність faucet для тестового ETH;
- Стабільна робота публічних RPC-ендпоінтів (Infura, Alchemy, ThirdWeb);
- Підтримка сканерів блоків (Optimism Sepolia Explorer).

Base Sepolia - тестова версія мережі Base, яку розвиває компанія Coinbase. Ця мережа також побудована на основі OP Stack і активно використовується для тестування додатків, які планується розгорнути в основній мережі Base. Переваги Base Sepolia включають:

- Високу стабільність та швидкість обробки транзакцій;
- Документацію від Coinbase Developers;
- Прості механізми отримання тестових токенів через faucet;
- Експлорери блоків із верифікацією контрактів.

Обидві мережі використовують Sepolia як базовий L1, що забезпечує сумісність між ними, дозволяє легко налаштовувати інструменти (зокрема Foundry) та пришвидшує інтеграцію з гаманцями (MetaMask, WalletConnect). Такий вибір забезпечує реалістичне тестове середовище, близьке до production-умов, що критично важливо для перевірки безпеки та ефективності смарт-контрактів перед основним деплойментом.

3.3.3 Переваги багатомережевого тестування

Використання одразу декількох тестових мереж на базі OP Stack - зокрема Optimism Sepolia та Base Sepolia - дає змогу забезпечити більш повну перевірку працездатності та стійкості смарт-контракту SuperChainERC20 у різних умовах.[19]

Це дозволяє провести порівняльний аналіз роботи контракту в різних OP Stack середовищах, зокрема за такими параметрами, як швидкість підтвердження транзакцій, вартість gas, стабільність RPC-запитів та якість інтеграції з блокчейн-експлорерами. Оскільки Optimism та Base мають власну інфраструктуру, це дозволяє імітувати поведінку контракту в умовах кількох незалежних мереж.

Багатомережевий підхід підвищує надійність і масштабованість розгортання. Тестування одного і того ж контракту в різних мережах дозволяє виявити проблеми, пов’язані з мереже-специфічними параметрами: різна реалізація sequencer, особливості кешування або обмеження на рівні faucet/деплойменту.

Така стратегія формує гнучку основу для подальшого production-деплойменту. Оскільки обидві мережі є повноцінними OP Stack-реалізаціями, код та інфраструктура, протестовані у Sepolia-мережах, можуть бути перенесені до Optimism Mainnet або Base Mainnet із мінімальними змінами. Це відповідає принципам повторюваності (reproducibility) і знижує ризик виникнення помилок при масштабуванні проєкту.[20]

У результаті багатомережеве тестування забезпечує більш високу якість підготовки до релізу смарт-контракту, дозволяє виявити граничні випадки поведінки системи та надає додаткову впевненість у стабільності реалізації.

4 РЕАЛІЗАЦІЯ ТОКЕНУ SUPERCHAINERC20

4.1 Підготовка середовища для розробки та тестування

4.1.1 Ініціалізація проєкту та структури

На початковому етапі було створено новий проєкт для розробки смарт-контракту SuperChainERC20 із використанням інструментарію Foundry - продуктивного фреймворку для Solidity. Цей інструмент забезпечує зручне середовище для компіляції, тестування та деплойменту смарт-контрактів у мережах, сумісних з Ethereum Virtual Machine (EVM).

Під час ініціалізації було автоматично згенеровано базову структуру проєкту, яка включає:

- каталог src/ для розміщення основних смарт-контрактів;
- папку lib/ для зовнішніх залежностей, зокрема бібліотек OpenZeppelin та forge-std;
- директорію test/ для написання тестів на Solidity;
- директорію script/ для розгортання та інтеракції з контрактами;
- конфігураційний файл foundry.toml з базовими параметрами проєкту.

Також було автоматично додано бібліотеку forge-std, яка використовується для створення тестів і відлагодження контрактів. Зображено на рисунку 4.1

```
root@Ubuntu:~# forge init superchain-erc20
Initializing /root/superchain-erc20...
Installing forge-std in /root/superchain-erc20/lib/forge-std (url: Some("https://github.com/foundry-rs/forge-std"), tag: None)
Cloning into '/root/superchain-erc20/lib/forge-std'...
remote: Enumerating objects: 2126, done.
remote: Counting objects: 100% (1008/1008), done.
remote: Compressing objects: 100% (131/131), done.
remote: Total 2126 (delta 932), reused 879 (delta 877), pack-reused 1118 (from 2)
Receiving objects: 100% (2126/2126), 720.75 KiB | 2.43 MiB/s, done.
Resolving deltas: 100% (1431/1431), done.
    Installed forge-std v1.9.7
    Initialized forge project
root@Ubuntu:~#
```

Рисунок 4.1 - Ініціалізація проєкту Foundry

Після цього середовище стало повністю готовим до конфігурації та інтеграції з цільовими мережами на базі OP Stack.

4.1.2 Налаштування конфігураційного файлу

Після створення структури проєкту було налаштовано файл foundry.toml, який визначає ключові параметри компіляції, підключення до зовнішніх бібліотек та конфігурацію мереж. Цей файл є основним конфігураційним центром для всієї розробки у межах середовища Foundry.

У секції profile.default було вказано такі параметри:

- шлях до вихідного коду (src) та директорії для скомпільованих артефактів (out);
- бібліотеки (lib), зокрема OpenZeppelin та forge-std;
- версія компілятора Solidity (0.8.19);
- включено оптимізацію та визначено кількість прогонів (optimizer_runs);
- відключено використання IR-компіляції (via_ir).

Особливу увагу було приділено remapping - це дозволяє імпортувати контракти бібліотек через логічні шляхи, що спрощує масштабування проєкту.

Також у конфігурації були додані RPC-ендпоінти для двох тестових мереж OP Stack:

- Optimism Sepolia;
- Base Sepolia.

Окрім цього, налаштовано параметри для верифікації контрактів у відповідних блокчейн-оглядачах (Etherscan, BaseScan) за допомогою API-ключів та вказання chain ID кожної мережі. Зображено на рисунку 4.2

foundry.toml

```toml
[profile.default]
src = "src"                    # Директорія з основними контрактами
out = "out"                    # Директорія для скомпільованих артефактів
libs = ["lib"]                 # Директорія з бібліотеками
solc_version = "0.8.19"        # Версія Solidity компайлера
optimizer = true               # Увімкнути оптимізацію газу
optimizer_runs = 200           # Кількість прогонів оптимізатора
via_ir = false                 # Не використовувати IR компіляцію
remappings = [
    "@openzeppelin/=lib/openzeppelin-contracts/"  # Маппінг для OpenZeppelin
]

# RPC ендпоінти для різних OP Stack мереж
[rpc_endpoints]
optimism_sepolia = "https://sepolia.optimism.io"      # Тестнет Optimism
base_sepolia = "https://sepolia.base.org"             # Тестнет Base
optimism = "https://mainnet.optimism.io"              # Основна мережа Optimism
base = "https://mainnet.base.org"                     # Основна мережа Base

# Налаштування для верифікації контрактів на блокчейн-оглядачах
[etherscan]
optimism_sepolia = {
    key = "${OPTIMISM_ETHERSCAN_API_KEY}",
    url = "https://api-sepolia-optimism.etherscan.io/api",
    chain = 11155420
}
base_sepolia = {
    key = "${BASESCAN_API_KEY}",
    url = "https://api-sepolia.basescan.org/api",
    chain = 84532
}
optimism = {
    key = "${OPTIMISM_ETHERSCAN_API_KEY}",
    url = "https://api-optimistic.etherscan.io/api",
    chain = 10
}
base = {
    key = "${BASESCAN_API_KEY}",
    url = "https://api.basescan.org/api",
    chain = 8453
}

# Налаштування для випадкового (fuzz) тестування
[fuzz]
runs = 1000                    # Кількість випадкових тестів
max_test_rejects = 65536       # Максимум відхилених тестів
seed = '0x3e8'                 # Початкове значення для генератора
dictionary_weight = 40         # Вага словника для генерації даних
include_storage = true         # Включити змінні пам'яті в тести
include_push_bytes = true      # Включити push байти в тести

# Налаштування форматування коду
[fmt]
line_length = 120              # Максимальна довжина рядка
tab_width = 4                  # Розмір табуляції
bracket_spacing = true         # Пробіли навколо дужок
int_types = "long"             # Використовувати повні назви типів
```

Рисунок 4.2 - Конфігураційний файл foundry.toml

Для перевірки коректності конфігурації було використано вбудовану функцію forge config, яка підтвердила правильне зчитування параметрів та відсутність синтаксичних помилок. Зображено на рисунку 4.3

```
root@Ubuntu:~/superchain-erc20# forge config --basic
[profile.default]
src = "src"
out = "out"
libs = ["lib"]
remappings = [
    "@openzeppelin/=lib/openzeppelin-contracts/",
    "forge-std/=lib/forge-std/src/",
    "openzeppelin-contracts/=lib/openzeppelin-contracts/",
]

# See more config options https://github.com/foundry-rs/foundry/blob/master/crates/config/README.md#all-options

root@Ubuntu:~/superchain-erc20#
```

Рисунок 4.3 - Результат перевірки конфігурації проєкту Foundry

Таким чином, конфігураційне середовище було успішно адаптоване під вимоги розгортання контрактів у мережах OP Stack, включно з підтримкою

зовнішніх бібліотек та верифікацією на Etherscan.

4.1.3 Створення env та підключення до мереж

Для забезпечення безпечного зберігання конфіденційних параметрів у межах розробки було створено файл .env, який містить усі необхідні змінні середовища. Такий підхід дозволяє уникнути прямого включення чутливої інформації (ключів, RPC-адрес, API-токенів) до вихідного коду, а також спрощує керування параметрами для різних середовищ - тестового та продуктивного. Зображено на рисунку 4.4

У вмісті .env було визначено такі ключові змінні:

- приватний ключ користувача, що використовується для підпису транзакцій;
- адреси RPC-провайдерів для Optimism Sepolia і Base Sepolia відповідно (через Alchemy);
- ключі для сервісів верифікації контрактів;
- назва токена, його символ і початковий обсяг емісії.

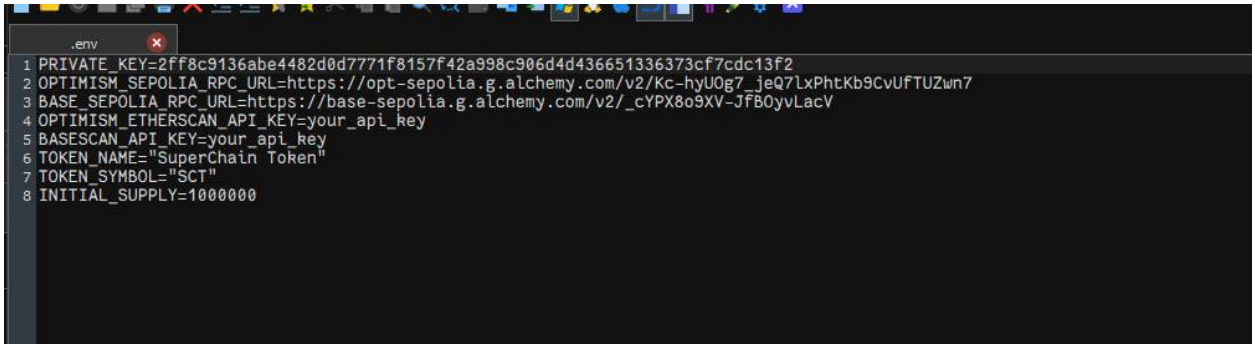

Рисунок 4.4 - Вміст .env файлу з основними параметрами проєкту

4.1.4 Отримання тестових токенів та перевірка балансу

Після налаштування змінних середовища у .env та підключення RPC-ендпоінтів до цільових тестових мереж виникла потреба поповнити гаманець

тестовими токенами ETH. Це необхідно для покриття вартості газу при компіляції, тестуванні й деплойменті смарт-контрактів.

Для цього було використано офіційний Superchain Faucet, який підтримує одночасну видачу токенів у мережах Optimism Sepolia та Base Sepolia. Сервіс дає змогу запитувати близько 0.025 ETH на кожну з мереж, один раз на 24 години на адресу. Зображено на рисунку 4.5

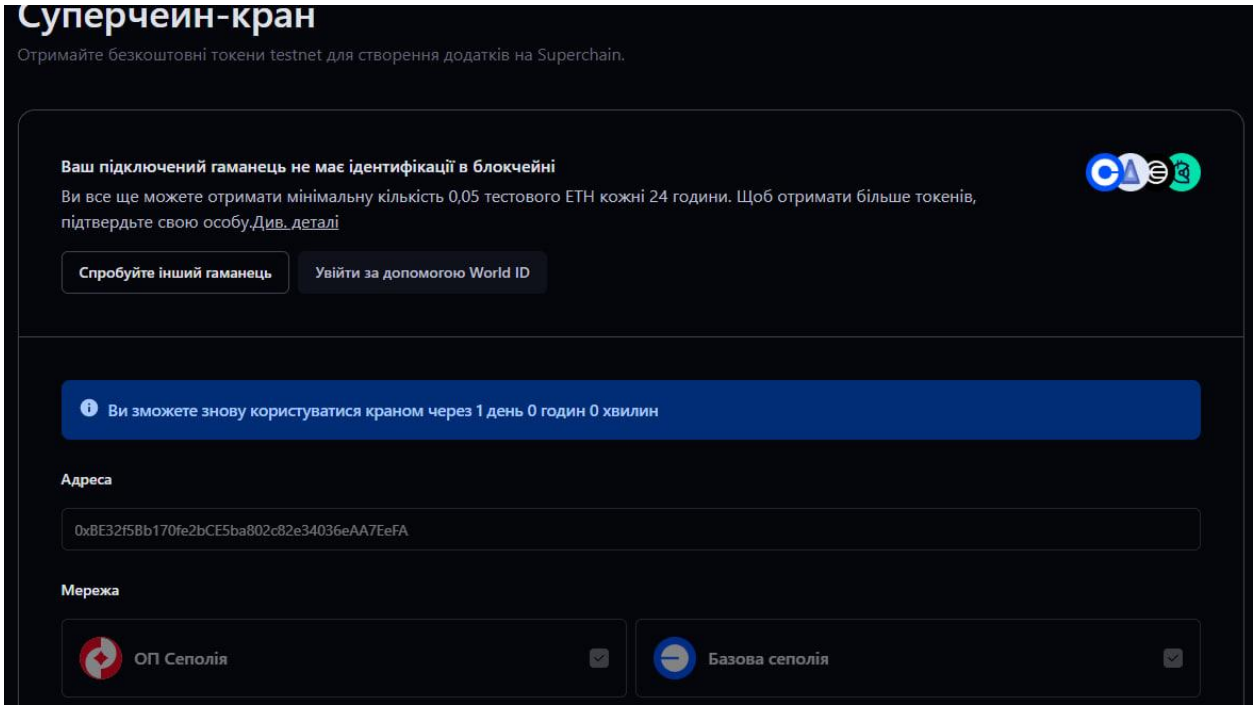

Рисунок 4.5 - Інтерфейс Superchain Faucet для OP Stack-мереж

Після отримання токенів було здійснено перевірку балансу гаманця в обох мережах. Для цього використано CLI-інструмент cast, який дозволяє перевірити стан рахунку за RPC-адресами. Зображено на рисунку 4.6

```
root@Ubuntu:~/superchain-erc20# cast balance $WALLET_ADDRESS --rpc-url $OPTIMISM_SEPOLIA_RPC_URL --ether
0.025000000000000000
root@Ubuntu:~/superchain-erc20# cast balance $WALLET_ADDRESS --rpc-url $BASE_SEPOLIA_RPC_URL --ether
0.025000000000000000
root@Ubuntu:~/superchain-erc20#
```

Рисунок 4.6 - Баланс тестового гаманця в Optimism Sepolia та Base Sepolia

Як видно з результатів, на момент перевірки гаманець мав по 0.025 ETH у кожній із тестових мереж, що є достатнім для проведення

деплойменту смарт-контракту та базового тестування взаємодій з мережею.

4.2 Розробка смарт-контракту SuperChainERC20

4.2.1 Архітектура контракту та спадкування

Контракт побудований на основі трьох базових компонентів бібліотеки OpenZeppelin:

- ERC20 стандарт для токенів із підтримкою балансу, переказів і дозволів;
- ERC20Burnable розширення, що дозволяє знищувати токени;
- Ownable шаблон для визначення власника контракту і контролю доступу до важливих функцій.

Таке поєднання дозволяє створити безпечний, розширюваний та керований токен із підтримкою мінімальних дефляційних механізмів.

4.2.2 Ключові параметри та ініціалізація

При розгортанні контракту передаються такі параметри:

- назва токена (name), відображається у гаманцях;
- коротка абревіатура (наприклад, SCT);
- загальна емісія при запуску;
- адреса власника (owner).

У тестовому проєкті для розгортання смарт-контракту були використані наступні параметри: назва токена - SuperChain Token, символ - SCT, початкова емісія становила 1 000 000 токенів, а власником контракту була вказана адреса розгортача, яка також отримала увесь початковий обсяг токенів.

Початкова кількість токенів автоматично масштабується з урахуванням

18 десяткових знаків, що є стандартом у мережі Ethereum.

4.2.3 Емісія та спалювання токенів

Контракт реалізує дві додаткові функції для управління пропозицією токенів:

- mint дозволяє власнику контракту створювати додаткові токени та надсилати їх на будь-яку адресу. Може використовуватись для винагород або розширення проєкту;
- burnFrom дозволяє зменшувати кількість токенів, що перебувають в обігу. Така функціональність може використовуватись для реалізації дефляційної політики.

Обидві функції обмежені модифікатором onlyOwner, що гарантує - доступ до них має виключно власник контракту.

4.2.4 Компіляція контракту та перевірка валідності

Компіляція контракту виконувалась у середовищі Foundry з такими налаштуваннями:

- Solidity версія: 0.8.20;
- Оптимізація: увімкнена з 200 прогонами;
- Час компіляції: ~782 мілісекунди;
- Кількість файлів: 31 (включно з бібліотеками).

На рисунку 4.7 зображено результат успішної компіляції

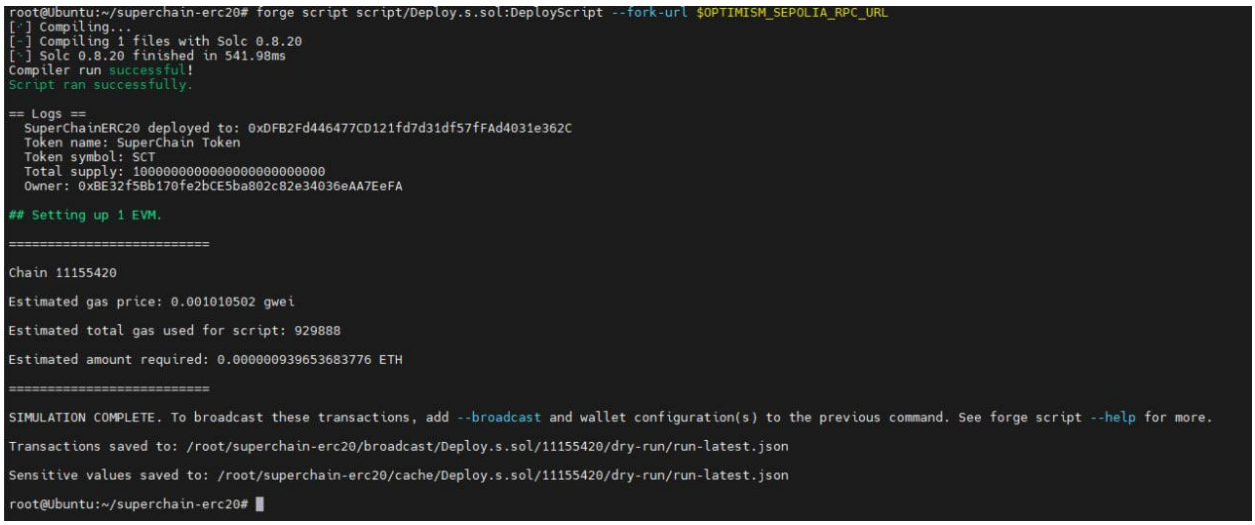

Рисунок 4.7 - Успішна компіляція контракту в середовищі Foundry

У процесі не виявлено жодних помилок або попереджень, що підтверджує правильність реалізації контракту. Це дозволяє перейти до наступного етапу - тестування функціональності та підготовки до деплою.

4.3 Тестування смарт-контракту SuperChainERC20

4.3.1 Тести та перевірка функціональності

З метою гарантування стабільної та правильної роботи смарт-контракту SuperChainERC20 було підготовлено комплексний набір тестів з використанням інструментарію фреймворку Foundry Test. Тестування здійснювалося безпосередньо на мові Solidity, що забезпечує максимальну наближеність до умов реального середовища виконання в блокчейні.

Тестовий контракт під назвою SuperChainERC20Test був розміщений у папці test/ та організований на основі базового класу, який наслідується від Test із бібліотеки forge-std. У межах тестового стенду були визначені три тестові адреси: owner (0x1) як повновладний власник контракту, а також user1 (0x2) і user2 (0x3) як звичайні користувачі для перевірки типових сценаріїв взаємодії.

На першому етапі було здійснено перевірку правильності початкової

ініціалізації контракту. Тест testInitialSetup() оцінював встановлення назви токена (SuperChain Token), символу (SCT), початкової емісії токенів відповідності балансу власника після деплою та правильності прив'язки адреси власника. Базову функцію переказу перевіряв тест testTransfer(), який моделював відправлення 1000 токенів від власника до user1 та перевіряв зменшення балансу відправника, збільшення балансу отримувача на очікувану кількість і незмінність загального обігу токенів.

Окремий акцент зроблено на тестуванні специфічних можливостей контракту - зокрема, функцій емісії та знищення токенів. Тест testMint() перевіряв право власника створювати додаткові токени, відповідність зростання балансу цільового користувача, коректне оновлення загальної емісії та збереження внутрішньої логіки контракту після операції.

Для перевірки функції burnFrom було змодельовано складніший сценарій у тесті testBurn(): власник спочатку переказував токени користувачу, який надавав дозвіл (approve) на витрачання певної кількості токенів, після чого власник виконував burnFrom, а система перевіряла зменшення як балансу, так і загального обігу токенів.

4.3.2 Перевірка безпеки та аналіз результатів

Особливе значення у процесі тестування приділялось безпековим аспектам, зокрема дотриманню доступу до функцій, обмежених модифікатором onlyOwner. Тест testMintOnlyOwner() передбачав спробу виклику функції mint зі сторони звичайного користувача (user1) з очікуванням її відхилення. Застосування vm.expectRevert() гарантувало, що тест буде успішним лише у випадку фактичної відмови системи виконувати несанкціоновану транзакцію.

Аналогічна перевірка була проведена для функції burnFrom, щоб підтвердити, що лише власник контракту має право знищувати токени з чужих рахунків.

Повний запуск тестів за допомогою команди forge test -vv продемонстрував успішне проходження всіх п’яти тестів: testBurn() - 63014 одиниць газу, testInitialSetup() - 29995, testMint() - 46997, testMintOnlyOwner() - 15217, testTransfer() - 45364 одиниць газу. Сукупний час виконання тестового пакету склав 8.67 мілісекунд, що свідчить про ефективність тестового середовища Foundry.

Базовий набір тестів охоплює всі ключові гілки логіки контракту: повне покриття конструктора та ініціалізації, повне тестування функцій mint і burnFrom, перевірка доступу через модифікатор onlyOwner та всіх базових ERC20-операцій. Такий рівень охоплення дозволяє з високим ступенем впевненості оцінити контракт як готовий до використання у продуктивному середовищі.

4.4 Деплой та верифікація контракту в мережах OP Stack

Для забезпечення відтворення та автоматизації процесу розгортання контракту було створено спеціалізований скрипт Deploy.s.sol, який розташовано в директорії script/. Цей скрипт використовує інструменти фреймворку Foundry Script для безпечного й налаштованого розгортання контракту в цільових блокчейн-мережах. Його архітектура базується на принципах читання параметрів із конфігураційного файлу .env, використанні функцій vm.envUint() і vm.envString() для надійного отримання змінних середовища, запуску vm.startBroadcast() для підписування транзакцій та логуванні основних даних про деплоймент.

Перед основним розгортанням контракту була виконана попередня симуляція з використанням параметра --fork-url, що дозволило протестувати логіку транзакцій без реального надсилання в мережу. Під час тестування в мережі Optimism Sepolia було зафіксовано приблизну ціну газу 0.00100502 gwei, витрату 929 888 одиниць газу, загальну вартість транзакції - 0.000000930536837776 ETH, а також Chain ID - 11155420.

Реальне розгортання в тестовій мережі Optimism Sepolia здійснювалось за допомогою параметра --broadcast. Під час цього було отримано такі результати: адреса контракту - 0xDFB2Fd446477CD121fd7d31df57fFAd4031e362C, хеш транзакції - 0x3441c564598475305456799b6a61151002a5d55f9de3c274afd1f22b75bafc, блок №29,243,708, фактичне споживання газу - 715 299 одиниць, вартість - близько $0.0007.

Такий самий процес був повторений у мережі Base Sepolia, що дозволило підтвердити сумісність рішення з кількома мережами. Результати розгортання в Base Sepolia: адреса контракту - 0xDFB2Fd446477CD121fd7d31df57fFAd4031e362C, хеш - 0x7f6204e1c048230320b5b63c4aba934624e37ce1e5f1c14cd08e08de23a86, блок №27,260,097, витрата газу - 715 299 одиниць, приблизна вартість - $0.0004. Варто відзначити, що в обох випадках контракт мав однакову адресу завдяки детермінованому характеру деплойменту.

Після успішного розміщення контракти стали доступні для перегляду та взаємодії через відповідні блокчейн-оглядачі. В обох випадках код є повністю відкритим і доступним для ознайомлення.

З метою демонстрації роботи додаткової функціональності було виконано тестову операцію mint у мережі Optimism Sepolia, в ході якої було створено 50 000 нових токенів. Транзакція завершилася успішно, що підтвердило коректну роботу модифікатора onlyOwner, правильну реалізацію функції емісії, збільшення загальної кількості токенів та оновлення балансу відповідної адреси.

На завершальному етапі було перевірено інтеграцію з існуючою інфраструктурою Web3. Токени було імпортовано в MetaMask через стандартний інтерфейс "Import tokens" із зазначенням адреси контракту - 0xDFB2Fd446477CD121fd7d31df57fFAd4031e362C. Після додавання токени відображались коректно, включно з балансом та можливістю здійснювати стандартні транзакції.

Процес розгортання та перевірки контракту SuperChainERC20 у мережах OP Stack був успішно завершений, що засвідчило технічну якість рішення, підтверджену використанням сучасних інструментів Foundry, підтримкою декількох мереж OP Stack, низькою загальною вартістю деплойменту (менше $0.002), повною функціональністю усіх заявлених можливостей і готовністю до інтеграції в екосистему Web3. У такий спосіб, ціль створення функціонального SuperChainERC20 токена з розширеним функціоналом управління на базі OP Stack була повністю реалізована.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи було реалізовано повноцінний токен SuperChainERC20 з розширеними можливостями управління в екосистемі OP Stack шляхом розробки, тестування та розгортання смарт-контракту в мережах Optimism Sepolia та Base Sepolia. Робота охопила всі ключові етапи життєвого циклу блокчейн-застосунку - від налаштування середовища розробки та проектування архітектури контракту до компіляції, тестування та верифікації в публічних тестових мережах.

У теоретичній частині було досліджено еволюцію платформи Ethereum від перших релізів до сучасних оновлень, архітектурні особливості Ethereum Virtual Machine та принципи роботи смарт-контрактів у децентралізованому середовищі. Детально вивчено технологію OP Stack як модульний фреймворк для створення масштабованих рішень другого рівня, концепцію Superchain та стандартизацію токенів через ERC-20 інтерфейс. Це дозволило обґрунтувати вибір технологічного стеку та архітектурного рішення для реалізації токену з функціями mint та burn, обмеженими правами власника.

У практичній частині було створено середовище розробки на базі фреймворку Foundry з підтримкою двох тестових мереж OP Stack. Розроблено смарт-контракт SuperChainERC20 з використанням бібліотек OpenZeppelin, що забезпечує стандартну ERC20 функціональність з додатковими можливостями управління емісією токенів. Створено комплексний набір тестів на мові Solidity, які забезпечують повне покриття функціональності контракту, включаючи перевірку механізмів безпеки та контролю доступу.

Після успішної компіляції контракт було розгорнуто в мережах Optimism Sepolia та Base Sepolia за ідентичними адресами (0xDFB2Fd446477CD121fd7d31df57fFAd4031e362C), що підтвердило детерміністичність процесу та правильність конфігурації. Проведене

тестування через CLI-інструменти та блокчейн-оглядачі підтвердило коректність усіх функцій контракту, включаючи створення додаткових токенів, їх знищення та інтеграцію з Web3-інфраструктурою через MetaMask.

Отримані результати свідчать про працездатність реалізованого токену SuperChainERC20, повну сумісність з екосистемою OP Stack та готовність до використання в продуктивному середовищі. Загальна вартість розгортання менше $0.002 демонструє економічну ефективність рішень другого рівня. Всі п'ять тестових сценаріїв пройшли успішно за 8.67 мілісекунд, що підтверджує як стабільність роботи контракту, так і ефективність обраного технологічного стеку для розробки multi-chain застосунків в екосистемі Superchain.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

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

2. Buterin V. Ethereum White Paper: A Next Generation Smart Contract and Decentralized Application Platform [Електронний ресурс]. – 2014. – Режим доступу: https://ethereum.org/en/whitepaper

3. Ethereum Foundation. Ethereum Documentation – The Merge, Rollups, and EVM [Електронний ресурс]. – 2024. – Режим доступу: https://ethereum.org

4. Optimism Foundation. OP Stack Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://stack.optimism.io

5. OpenZeppelin. Contracts Library (v5.0) [Електронний ресурс]. – 2024. – Режим доступу: https://docs.openzeppelin.com/contracts

6. GitHub – Optimism Monorepo [Електронний ресурс]. – Режим доступу: https://github.com/ethereum-optimism/optimism

7. Ethereum Foundation. Ethereum Improvement Proposals (EIPs) [Електронний ресурс]. – Режим доступу: https://eips.ethereum.org

8. Certik Foundation. Smart Contract Security Best Practices [Електронний ресурс]. – 2023. – Режим доступу: https://www.certik.com/resources

9. Alchemy. Deploying Smart Contracts to Optimism [Електронний ресурс]. – 2024. – Режим доступу: https://docs.alchemy.com/docs/optimism

10. Polygon Labs. zkEVM, Rollup Architectures and Modular Blockchains [Електронний ресурс]. – 2024. – Режим доступу: https://polygon.technology

11. Foundry Book. Foundry for Ethereum Application Development [Електронний ресурс]. – Режим доступу: https://book.getfoundry.sh

12. Base Docs. Base Sepolia Testnet – Developer Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://docs.base.org

13. Paradigm. Foundry Reference – Compilation and Testing [Електронний

ресурс]. – Режим доступу: https://book.getfoundry.sh/reference

14. Solidity Docs. Solidity 0.8.20 Release Notes [Електронний ресурс]. – 2023. – Режим доступу: https://docs.soliditylang.org

15. Superchain. About Superchain Architecture [Електронний ресурс]. – 2024. – Режим доступу: https://www.optimism.io/superchain

16. Coinbase. Introduction to Superchain Ecosystem [Електронний ресурс]. – 2024. – Режим доступу: https://www.coinbase.com/blog/what-is-the-superchain

17. Chainlist. Ethereum RPC Endpoints and Network Data [Електронний ресурс]. – Режим доступу: https://chainlist.org

18. Forge Std Library. Standard Library for Foundry Testing [Електронний ресурс]. – Режим доступу: https://github.com/foundry-rs/forge-std

19. Optimism. Faucet for Optimism Sepolia [Електронний ресурс]. – Режим доступу: https://community.optimism.io/docs/useful-tools/faucets

20. OpenZeppelin. Ownable Contract Documentation [Електронний ресурс]. – Режим доступу: https://docs.openzeppelin.com/contracts/5.x/access#Ownable

21. Дімітров М.М., Сітніков В.І. Інтелектуальні вимірювальні системи для моніторингу довкілля: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.1: секції 1,5. Баку: ІСУ АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. С. 141.