Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів. Аналіз підходів для реалізації смарт контрактів на мові TypeScript .

(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-3

Артем ОФАТЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Сергій МАР`ЇН
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри ____________ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _________
(підпис)
«____» __________ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Офатенку Артему Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів. Аналіз підходів для реалізації смарт контрактів на мові TypeScript»

Затверджена наказом по університету від 15.04. 2025р. № 290 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13.06.2025

3. Вихідні дані до роботи *календарний план виконання роботи, методичні вказівки, технології Typescript, JavaScript, Solidity, використання деяких ресурсів з інтернету за обраною темою, пояснювальна записка, перелік методів за критеріями.*

4. Перелік питань, що потрібно опрацювати в роботі
вивчення можливостей предметної області, вирішення завдань предметної області, проведення компіляції мови програмування Typescript в Solidity, проведення ексепереметиального порівняння ефективнфості з метою покращтити оптимізацію, проведення аналізу джерел та проведення теоретичного та практичного дослідження і сфері смарт-контрактів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	11.04.2025	Виконано
2	Аналіз предметної галузі і постановка задачі	15.04.2025	Виконано
3	Теоретичне дослідження	21.04.2025	Виконано
4	Практичне дослідження	01.05.2025	Виконано
5	Підготовка до апробації результатів дослідження. Публікація матеріалів	08.05.2025	Виконано
6	Підготовка пояснювальної записки	10.05.2025	Виконано
7	Підготовка презентації та доповіді	12.05.2025	Виконано
8	Перевірка на плагіат	09.06.2025	Виконано
9	Нормоконтроль	10.06.2025	Виконано
10	Рецензування	11.06.2025	Виконано
11	Попередній захист	12.06.2025	Виконано
12	Занесення диплома в електронний архів	12.06.2025	Виконано
13	Допуск до захисту у зав. кафедри	13.06.2025	Виконано

Дата видачі завдання __11 квітня__ 2025р.

Здобувач ________________ Артем ОФАТЕНКО
(підпис)

Керівник роботи ________________________ доц. Сергій МАР`ЇН
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка складається: 87с., 5 рис., 2 табл., 11 джерел.

ДЖАВА-СКРІПТ, BLOCKCHAIN, COMPILATION, ETHERIUM, EVM, SOLIDITY, TYPESCRIPT.

Об'єктом дослідження виступає мова програмування TypeScript та її використання для створення смарт-контрактів у блокчейн-системах, зокрема шляхом трансляції програм у мову Solidity.

Метою роботи є дослідження можливостей TypeScript для розробки смарт-контрактів, аналіз існуючих засобів трансляції та розробка рекомендацій щодо вдосконалення процесу компіляції та інструментарію.

Методи дослідження включають аналіз літературних джерел і технічної документації, практичне тестування наявних інструментів, моделювання процесу трансляції, а також порівняльний аналіз синтаксису та семантики TypeScript і Solidity.

У ході дослідження було виявлено переваги використання TypeScript для створення смарт-контрактів, зокрема зручність статичної типізації, сучасні засоби розробки, а також інтеграцію з екосистемою JavaScript і Node.js. Було зазначено, що існуючі інструменти суттєво спрощують автоматизацію генерації типів для взаємодії з контрактами Solidity. Запропоновано вдосконалення інструментів шляхом деяких методів. Оптимізації процесу перетворення TypeScript у Solidity з урахуванням особливостей мов. Покращення можливостей тестування смарт-контрактів у середовищі TypeScript.

Створення нових модулів для перевірки правильності коду перед його трансляцією в Solidity

Результати дослідження підтверджують перспективність використання TypeScript для розробки та підтримки смарт-контрактів, а також наголошують на необхідності подальшого вдосконалення інструментів, що забезпечують інтеграцію TypeScript із блокчейн-розробкою.

ETHERIUM, TYPESCRIPT, JS, COMPILATION, EVM, SOLIDITY, BLOCKCHAIN.

The object of research is the TypeScript programming language and its use for creating smart contracts in blockchain systems, in particular by translating programs into the Solidity language.

The purpose of the work is to explore the capabilities of TypeScript for developing smart contracts, analyze existing translation tools, and develop recommendations for improving the compilation process and tools.

Research methods include analysis of literary sources and technical documentation, practical testing of available tools, modeling of the translation process, and comparative analysis of the syntax and semantics of TypeScript and Solidity.

The study identified the advantages of using TypeScript for creating smart contracts, including the convenience of static typing, modern development tools, and integration with the JavaScript and Node.js ecosystem. It was noted that existing tools significantly simplify the automation of type generation for interaction with Solidity contracts. At the same time, certain shortcomings were identified, including difficulties in working with specific Solidity constructs and insufficient support for individual compilation mechanisms.

It is proposed to improve the tools by some methods. Optimization of the process of converting TypeScript to Solidity taking into account the peculiarities of the languages. Improvement of the capabilities of testing smart contracts in the TypeScript environment.

Creation of new modules for checking the correctness of the code before translating it into Solidity

The results of the study confirm the promising use of TypeScript for developing and supporting smart contracts, and also emphasize the need for further improvement of tools that ensure the integration of TypeScript with blockchain development.

Завідувачу кафедри
ПІ__________________
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я,__________________Офатенко Артем Олегович_______________________
(прізвище, ім'я, по батькові)

здобувач вищої освіти на другому (магістерському) рівні вищої освіти
академічної групи___ІПЗм-23-3____

кафедра__________________ програмної інженерії __________________________,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему «Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів. Аналіз підходів для реалізації смарт контрактів на мові TypeScript»_______________________________,
(назва роботи)

що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата Підпис

ЗМІСТ

Вступ……………………………………………………………………………9
1 Аналіз предметної галузі………………………………………………….... 11
1.1 Важливість та актуальність вибору дослідження………………………..11
1.1.1 Оцінка поточних тенденцій та майбутніх перспектив………………...12
1.1.2 Тема дослідження: Технологія Блокчейну та смарт-контракти………14
1.1.3 Області використання смарт-контрактів……………………………….15
1.2 Програмні мови для смарт-контрактів……………………………………..16
1.3 Детальне роз'яснення та огляд мови Typescript……………………………19
2 Огляд й аналіз літературних, наукових джерел……………………………...23
2.1 Огляд сучасних досліджень і джерел………………………………………23
2.2 Література та аналіз…………………………………………………………25
2.3 Новизна та актуальність…………………………………………………….. 26
2.4 Висновки по літературі………………………………………………………27
3 Постановка задачі…………………………………………………….............. 29
3.1 Огляд списку використаних технологій…………………………………… 30
4 Теоретичне дослідження………………………………………………………32
4.1 Основи створення та розробки смарт-контрактів…………………………32
4.1.1 Основні принципи функціонування смарт-контрактів у блокчейн-мережах……………………………………………………………………….... 33
4.1.2 Вивчення прогалини у Solidity та TypeScript……………………………..34
4.1.3 Виклики та різниця в семантиці…………………………………………...35
4.1.4 Рішення та роль проміжного представлення (IR)…………………….. 36
4.2 Проведення паралельної компіляції у сфері TypeScript та Solidity……...38
4.3 Масштабованість……………………………………………………………...39
4.4 Мета теоретичного дослідження та підсумки……………………………... 41
4.5 Компіляція TypeScript в Solidity використовуючи ts-morhp……………...42
5 Практичне дослідження………………………………………………………45
5.1 Проведення інтеграції під час розробки смарт-контрактів………………..45

5.1.1 TypeScript та вза’ємодія з Web3…………………………………...45
5.1.2 Компіляція в Solidity……………………………………………..48
5.1.3 Проведення перевірки та підготовки даних…………………………….53
5.2 Методи порівння за критерієм швидкості компіляції…………………… 59
5.3 Методи порівння за критерієм безпечности у використанні…………….. 61
5.4 Методи роботи паралельної компіляції файлів typescript у Solidity …….63
Висновки……………………………………………………………………67
Перелік джерел посилання………………………………………………… 72
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії……………………………………….. 73
Додаток А……………………………………………………………………74
Додаток Б…………………………………………………………........ 78
Додаток Г…………………………………………………………………… 80
Додаток Д……………………………………………………………………85
Додаток Е…………………………………………………………………… 86
Додаток Ж……………………………………………………………………87

ВСТУП

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів відкриває нові горизонти в автоматизації виконання угод у децентралізованих мережах. TypeScript надає потужні інструменти для створення смарт-контрактів, які є важливими складовими децентралізованих додатків (dApps). Ці контракти можуть бути використані в різних сферах, таких як фінансові послуги (DeFi), платформи для голосування, NFT-маркетплейси та управління ланцюгами постачання, що підкреслює їх значення в сучасній цифровій економіці. Аналіз підходів до реалізації смарт-контрактів на TypeScript дозволяє виявити переваги та виклики, пов'язані з їх використанням.

Розробка смарт-контрактів зазвичай виконується за допомогою спеціалізованих мов програмування, таких як Solidity, Vyper (для Ethereum) або Rust (для блокчейнів Solana, Near, Polkadot). Проте обмежені можливості цих мов та високі вимоги до безпеки спонукають розробників шукати більш універсальні та зручні інструменти, такі як TypeScript, для створення смарт-контрактів. Дослідження підходів до реалізації смарт-контрактів на TypeScript може відкрити нові перспективи в цій галузі.

TypeScript, типізоване розширення JavaScript є однією з перспективних мов програмування для розробки. Вона здобула популярність у веб-розробці завдяки своїй універсальності, розвиненій системі типізації та сумісності з сучасними фреймворками, такими як React, Angular і Vue.js. Завдяки потужній екосистемі TypeScript поступово починає займати своє місце в розробці смарт-контрактів, незважаючи на те, що спочатку не була створена для цієї сфери. Дослідження можливостей TypeScript у контексті смарт-контрактів відкриває нові горизонти для розробників.

TypeScript не має вбудованої підтримки для розробки смарт-контрактів, проте існує безліч інструментів і бібліотек, таких як Ether.js, Web3.js, Hardhat, TypeChain і Thirdweb SDK. Ці інструменти дозволяють взаємодіяти з блокчейном, спрощують процес розробки та навіть надають можливість генерувати код на Solidity. Завдяки цим можливостям для розробників відкриваються нові

горизонти, а доступ до блокчейн-екосистеми стає значно простішим. Дослідження підходів до реалізації смарт-контрактів на TypeScript може суттєво полегшити цю задачу.

Актуальність обраної теми обумовлена кількома ключовими факторами Зростаюча популярність TypeScript стає все більш популярним серед розробників завдяки своїй статичній типізації, що дозволяє виявляти помилки на етапі написання коду. Це може суттєво підвищити якість коду смарт-контрактів. Потреба в нових інструментах , оскільки блокчейн-технології продовжують розвиватися, існує потреба в нових інструментах і мовах програмування, які можуть спростити процес розробки смарт-контрактів. Дослідження TypeScript як альтернативи може відкрити нові можливості для розробників. Спрощення розробки аналізу підходів до реалізації смарт-контрактів на TypeScript може виявити способи спростити процес розробки, зменшити час на навчання та підвищити продуктивність розробників, які вже знайомі з цією мовою. Інтеграція з існуючими екосистемами TypeScript може бути інтегрований з популярними бібліотеками та фреймворками, такими як Angular, React, Web3.js та Ethers.js, що дозволяє створювати більш ефективні та зручні інтерфейси для взаємодії з блокчейном. Безпека смарт-контрактів також, оскільки безпека є критично важливою у сфері блокчейн-технологій , у якості інформації про даний тип технології викорстовуючи загальне джерело [1], можливості статичної типізації TypeScript можуть допомогти виявляти потенційні вразливості на етапі написання коду, що зменшить ризики, пов'язані з експлуатацією смарт-контрактів. Таким чином, дослідження можливостей TypeScript для розробки смарт-контрактів є важливим кроком у напрямку вдосконалення інструментів для розробників і підвищення ефективності розробки в блокчейн-екосистем.

Таким чином, мета дослідження полягає в глибокому аналізі можливостей TypeScript у контексті розробки смарт-контрактів, що дозволить виявити нові перспективи для розробників і сприяти розвитку інновацій у цій галузі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Важливість та актуальність вибору дослідження

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів є надзвичайно важливим і актуальним з кількох причин.

Зростання популярності блокчейн-технологій у останні роки блокчейн-технології стали основою для численних інновацій у фінансовій сфері, управлінні даними та інших галузях. Смарт-контракти, як невід'ємна частина цієї технології, набувають все більшої популярності. Вивчення нових мов програмування, таких як TypeScript, може сприяти розширенню можливостей розробки смарт-контрактів. Потреба в нових інструментах, розробка смарт-контрактів традиційно здійснюється за допомогою мов, таких як Solidity. Однак, зростаюча кількість розробників, які вже мають досвід у JavaScript/TypeScript про яке детальніше порушується питання у другому джерелі [2], створює потребу в інструментах та методах, які дозволяють використовувати знайомі мови для написання смарт-контрактів. Інтеграція з сучасними фреймворками TypeScript добре інтегрується з популярними фреймворками для фронтенд-розробки, такими як Angular та React. Це відкриває нові можливості для створення децентралізованих застосунків (dApps), що потребують взаємодії з смарт-контрактами.

Дослідження в цій галузі може стати основою для подальших наукових робіт, що вивчають нові технології в контексті блокчейн-розробки. Це може допомогти створити нові стандарти та практики для розробників. Безпека та надійність смарт-контрактів часто використовуються в фінансових операціях, тому питання безпеки є критично важливим. Дослідження можливостей TypeScript може виявити нові підходи до забезпечення безпеки, такі як статичний аналіз коду і типо-безпечність, що дозволить зменшити ризики.

Потреба в нових інструментах також зростає у вигляді розробці смарт-контрактів традиційно здійснюється за допомогою мов, таких як Solidity. Однак, зростаюча кількість розробників, які вже мають досвід у JavaScript/TypeScript, створює потребу в інструментах та методах, які дозволяють використовувати знайомі мови для написання смарт-контрактів.

1.1.1 Оцінка поточних тенденцій та майбутніх перспектив

Зростання популярності TypeScript стає все більш популярним серед розробників завдяки своїй статичній типізації, що забезпечує більшу надійність коду та легкість у його читанні. Ця мова активно використовується у веб-розробці, що створює сприятливе середовище для її застосування в розробці смарт-контрактів. Розробники шукають способи інтеграції знайомих технологій у нові сфери. TypeScript, завдяки своїй сумісності з JavaScript, може стати мостом між традиційною веб-розробкою та новими децентралізованими технологіями. В умовах зростаючої кількості атак на смарт-контракти, безпека стає пріоритетом. TypeScript може запропонувати механізми для статичного аналізу коду, що допоможе виявляти потенційні вразливості ще на етапі розробки. З'являються нові інструменти та бібліотеки, які полегшують розробку смарт-контрактів на TypeScript.

Наприклад, бібліотеки для роботи з Ethereum, такі як ethers.js, активно підтримують TypeScript, що робить його ще більш привабливим для розробників. Блокчейн-індустрія швидко розвивається, і нові стандарти, такі як ERC-20 або ERC-721, вимагають від розробників адаптації. TypeScript може забезпечити зручний інтерфейс для реалізації цих стандартів, що полегшить їх використання. TypeScript є надмножиною JavaScript - рисунок 1, що означає, що будь-який код JavaScript є дійсним кодом TypeScript. Це забезпечує плавний перехід для розробників, які вже мають досвід у JavaScript, і дозволяє їм поступово впроваджувати статичну типізацію в існуючі проекти. У контексті смарт-контрактів це означає, що розробники можуть використовувати знайомі їм бібліотеки та фреймворки, такі як Web3.js або Ethers.js, без необхідності повного переписування коду.

Розширення застосування TypeScript у блокчейн-розробці надає популярності TypeScript як можна очікувати, тому більше проектів почне використовувати цю мову для розробки смарт-контрактів, що призведе до появи нових бібліотек та фреймворків. Очікується розвиток інструментів для статичного

аналізу та тестування смарт-контрактів, що дозволить розробникам виявляти помилки та вразливості ще на етапі написання код.

Рисунок 1 – Логотип JavaScript(https://uk.javascript.info/)

TypeScript може стати основою для нових технологій у сфері децентралізованих фінансів (DeFi), NFT та інших інноваційних рішень, що з'являються на ринку. З розвитком технологій, пов'язаних із блокчейном та TypeScript, зростатиме попит на фахівців, які володіють цими технологіями. Це може призвести до нових освітніх програм та курсів, що спеціалізуються на цій темі. TypeScript може забезпечити кращу інтеграцію між смарт-контрактами та традиційними веб-додатками, що дозволить створювати більш комплексні та зручні у використанні децентралізовані застосунки.

TypeScript має потенціал стати важливим інструментом для створення крос-платформених смарт-контрактів, які можуть працювати на різних блокчейнах, що стає все більш необхідним через розвиток нових платформ. Його інтеграція з формальними методами верифікації, такими як моделювання станів і доведення теорем, може значно підвищити рівень безпеки смарт-контрактів. Крім того, вдосконалення інструментів для тестування та налагодження, включаючи інтегровані середовища розробки, дебагери та фреймворки для тестування, є важливим напрямком для подальшого розвитку. У підсумку, дослідження можливостей TypeScript для розробки смарт-контрактів не лише актуальне сьогодні, але й має великі перспективи на майбутнє. З розвитком технологій та

зростанням інтересу до децентралізованих рішень, TypeScript може стати важливим інструментом

1.1.2 Тема дослідження: Технологія Блокчейну та смарт-контракти

Блокчейн являє собою дистрибутивну базу даних, яка має структуру ланцюга блоків, в якості приклада для малюванні подібних схем використовуючи данні с джерела [3]. Кожен блок містить набір транзакцій, хеш попереднього блоку, тимчасову мітку (timestamp) і додаткові метадані, які забезпечують його унікальність та з'єднання з іншими блоками. Завдяки такій організації блокчейн утворює непорушний ланцюг даних, що захищений від фальсифікацій та несанкціонованих змін. Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів зосереджується на аналізі її переваг у порівнянні з традиційними мовами, такими як Solidity. Основною метою є вивчення підходів до реалізації смарт-контрактів на TypeScript, а також оцінка інструментів і бібліотек, які підтримують цей процес. Методологія включає аналіз літератури, емпіричні дослідження з розробки тестових смарт-контрактів, опитування експертів та оцінку безпеки. Очікується, що результати дослідження виявлять ключові переваги TypeScript, такі як статична типізація, що покращує надійність та безпеку коду, а також виявлять потенційні ризики, пов'язані з його використанням. Це дослідження має на меті надати розробникам нові інструменти для створення більш надійних децентралізованих додатків, що може суттєво вплинути на розвиток технології блокчейн.

У контексті дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів особливу увагу слід приділити питанням безпеки. Використання TypeScript, завдяки його статичній типізації, може суттєво знизити ймовірність виникнення помилок у коді, що є критично важливим у середовищі, де навіть незначна уразливість може призвести до значних фінансових втрат. Аналіз підходів до реалізації смарт-контрактів на TypeScript включає вивчення існуючих бібліотек та інструментів, таких як Hardhat або Truffle, які підтримують TypeScript, а також методик тестування та перевірки безпеки смарт-контрактів.

Оцінка потенційних уразливостей, таких як переповнення буфера або невірна обробка даних, допоможе виявити слабкі місця в реалізації контрактів. Дослідження також передбачає збір думок експертів у галузі безпеки блокчейн-технологій, що дозволить створити комплексну картину щодо застосування TypeScript у розробці смарт-контрактів, підкреслюючи його переваги та недоліки в контексті безпеки.

1.1.3 Області використання смарт-контрактів

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів охоплює кілька ключових областей використання, які демонструють потенціал цієї технології у різних сферах. По-перше, смарт-контракти можуть бути ефективно використані у фінансових послугах, зокрема для автоматизації процесів кредитування, страхування та торгівлі, що забезпечує прозорість і зменшує ризики шахрайства. По-друге, вони знаходять застосування в управлінні ланцюгами постачання, де смарт-контракти можуть автоматизувати виконання угод між постачальниками та замовниками, забезпечуючи відстеження товарів у реальному часі. Третя область - це децентралізовані фінансові додатки (DeFi), які використовують смарт-контракти для створення нових фінансових інструментів, таких як децентралізовані біржі та стейблкоїни.

Крім того, смарт-контракти можуть бути корисними в сфері управління цифровими активами, зокрема для токенізації нерухомості, мистецтва або інших цінностей, що дозволяє спростити процеси купівлі-продажу. Вони також мають потенціал у сфері голосування, де можуть забезпечити прозорість та недоторканність виборів. Важливо зазначити, що використання TypeScript у цих областях може підвищити надійність і безпеку розроблених смарт-контрактів завдяки його статичній типізації та підтримці сучасних інструментів розробки. Таким чином, дослідження TypeScript у контексті смарт-контрактів відкриває нові горизонти для інновацій у різних галузях, сприяючи розвитку децентралізованих технологій. Створення організації такої як ДАО(Децентралізованих Автономних

Організацій). Управління через смарт-контракти дозволяє учасникам організації голосувати за зміни в протоколах, розподіл ресурсів або вибір керівництва.

NFT-Маркетплейси (Цифрові Активи). Невзаємозамінні токени (NFT) - це унікальні цифрові об'єкти, які мають можливість провести операцію підтвердження та право власності на різноманітний цифровий контент, такий як зображення, музика чи відео. Смарт-контракти є основним елементом NFT-платформ,при цьому автоматизуючи процеси купівлі та продажу.

Проведення моменту з продажу та аукціону. Смарт-контракти мають нагоду та забезпечують автоматичне проведення аукціонів у реальному часі, при цьому фіксуючи ставки та визначаючи переможців.

1.2 Програмні мови для розробки смарт-контрактів

Solidity – це мова програмування, орієнтована на контракти, яка була створена для розробки смарт-контрактів на блокчейні Ethereum та інших сумісних з EVM платформах. Вона поєднує синтаксис і концепції з таких мов, як C++, Python і JavaScript, що забезпечує її гнучкість і функціональність при створенні децентралізованих додатків (dApps).

Основне призначення Solidity полягає у створенні смарт-контрактів, здатних працювати в середовищі Ethereum Virtual Machine (EVM). Головною метою є автоматизація транзакцій та зберігання даних в незмінному вигляді на блокчейні.Найбільш поширена мова Solidity для написання смарт-контрактів на платформі Ethereum. Вона спеціалізується на створенні децентралізованих додатків (dApps) і має велику спільноту та підтримку.

Solidity володіє рядом потужних функцій, які дозволяють смарт-контрактам ефективно працювати в екосистемі Ethereum. Контракти в Solidity виконують роль основних структур, що включають змінні стану, функції та події. Вони визначають правила взаємодії між учасниками мережі. Функції можуть викликатися як зсередини контракту, так і ззовні. Для виклику функції зсередини використовують її ім’я, а для виклику ззовні - через транзакцію або виклик з іншого контракту. Функції можуть використовувати модифікатори для

додаткових умов перед виконанням. Наприклад, модифікатор для перевірки, чи є виконавець контракту власником.

Приклад функції:

```
// приклад функції у Solidity
pragma solidity ^0.8.0;

contract SimpleContract {
    string public message;

    constructor(string memory initialMessage) {
        message = initialMessage;
    }

    function updateMessage(string memory newMessage) public {
        message = newMessage;
    }
}
```

Можливість проведення підтримки функцій та деяких подій функції які є елементами, що описують дії, які можуть виконуватись всередині контракту. Вони можуть мати різні рівні доступу, бути публічними або приватними. Події в свою чергу дають можливість реєструвати важливі події в блокчейні, що дає змогу відслідковувати їх у майбутньому

Наданий приклад події:

```
event FundsDeposited(address indexed from, uint amount);
```

Модифікатори надають можливість проводити контроль до доступу та функцій контракту. Як приклад, деякі функції може виконувати тільки користувач – власник контракту. У Solidity модифікатори - це спеціальні функції, які використовуються для зміни поведінки інших функцій. Вони дозволяють додавати умови або логіку перед виконанням функції, що робить код більш організованим і безпечним. Модифікатори часто використовуються для реалізації перевірок доступу, обмежень або для виконання певних дій.

Приклад модифікатора доступу:

```
modifier onlyOwner() {
    require(msg.sender == owner, "You are not the owner!");
    _;
}
```

Solidity надає підтримку для різних типів даних, включаючи масиви, мапи і структури (struct). У Solidity struct (структура) - це складний тип даних, який дозволяє групувати кілька змінних різних типів під одним ім'ям. Це особливо корисно для організації даних у смарт-контрактах, оскільки дозволяє створювати більш складні структури, які можуть представляти об'єкти з кількома атрибутами.

Приклад структури даних:

```
struct User {
    string user;
    uint pass;
}
User[] public users;
```

Функції в Solidity є критично важливими для визначення логіки смарт-контрактів. Вони дозволяють виконувати різні операції, взаємодіяти з даними та забезпечувати безпеку через контроль доступу. Розуміння функцій є основою для ефективної розробки смарт-контрактів.

У Solidity, мові програмування для смарт-контрактів на платформі Ethereum, функції є основними елементами, що виконують логіку контракту. Вони оголошуються за допомогою ключового слова function, після чого вказується ім’я, параметри та тип повертаємого значення. Функції можуть мати різні рівні видимості, такі як public, private, internal і external, а також статуси, як-от view, pure і payable, які визначають їхні можливості щодо читання або зміни стану блокчейну. Параметри можуть мати різні типи даних, і функції можуть повертати значення, що вказується після returns. Вони можуть викликатися як зсередини контракту, так і ззовні, наприклад, через транзакції або виклики з інших контрактів. Модифікатори дозволяють додавати умови перед виконанням функцій, забезпечуючи контроль доступу. Таким чином, функції в Solidity є критично важливими для реалізації логіки смарт-контрактів, взаємодії з даними та забезпечення безпеки.

Solidity, хоча й популярна для розробки смарт-контрактів, має кілька суттєвих недоліків. По-перше, вона схильна до помилок, таких як переповнення змінних, що може призвести до фінансових втрат. По-друге, обмежена підтримка типів даних ускладнює роботу з комплексними структурами. Складний синтаксис може бути важким для новачків, а високі витрати на газ роблять деякі операції дорогими. Відсутність стандартизації та непередбачувані зміни в оновленнях мови також створюють труднощі для розробників. Всі ці фактори вказують на те, що, незважаючи на свої переваги, Solidity має значні обмеження, які можуть ускладнити створення безпечних смарт-контрактів.

1.3 Детальне роз'яснення та огляд мови Typescript

TypeScript - це розширення для JavaScript, яке додає статичну типізацію та інші можливості, що значно підвищують якість коду та ефективність розробки. Цю мову активно використовують у веб-розробці для створення як фронтенд-, так і бекенд-рішень. Завдяки перевагам у типізації, об'єктно-орієнтованому підходу та розвиненій екосистемі, TypeScript набирає все більшої популярності в області блокчейн-розробки, зокрема для розробки смарт-контрактів та інтеграції з різними блокчейн-мережами.

Один із основних аспектів TypeScript - це статична типізація, яка відсутня в JavaScript. Вона дає можливість визначати типи для змінних, параметрів функцій та їх результатів ще на етапі компіляції, що дозволяє виявляти помилки на ранньому етапі, до того як код буде виконано. Завдяки типізації змінних TypeScript допомагає уникнути поширених помилок, таких як несанкціоновані операції з рядками або неправильна робота з масивами та об'єктами.

Наведемо приклад:

```
let amount: number = 100;  // Маємо визначення, amount - це число
amount = "string";   // Виведення помилки компіляції, тип буде несумісним
```

Визначення типів для параметрів та результатів функцій допомагає точніше зрозуміти, які саме дані обробляються функцією, а також покращує інтеграцію з іншими компонентами програми

Наведемо приклад:

```
function calculateSum(c: number, d: number): number {
    return c + d;
}

calculateSum(15, 25); // Буде працювати без помилок
calculateSum(15, "25"); // Видасть помилку компіляції
```

Це значно зменшує ймовірність помилок, пов'язаних з неправильним використанням даних, що є критично важливим при розробці смарт-контрактів, де точність виконання операцій має велике значення.

Об’єктно-орієнтоване програмування: класи, інтерфейси, дженерики TypeScript підтримує важливі принципи об’єктно-орієнтованого програмування (ООП), що сприяє більш організованому та зрозумілому коду. Це дає можливість розробникам використовувати перевірені шаблони проектування для створення ефективних і підтримуваних рішень.

Наявність класів: TypeScript дозволяє створювати класи, які використовуються для формування об’єктів та організації коду у вигляді окремих модулів. Вони можуть містити конструкції, методи й властивості.

```
class Contract {
    address: string;
    constructor(address: string) {
        this.address = address;
    }

    displayAddress(): void {
        console.log(`Contract Address: ${this.address}`);
    }
}

let contract = new Contract("0x1234567890abcdef");
contract.displayAddress(); // Буде виводити адресу контракту
```

Інтерфейс завжди виступ однією с головних можливостей функціоналу.Інтерфейси дають змогу встановлювати структуру для об'єктів або класів без їх фактичної реалізації, створюючи своєрідний контракт для роботи з інформацією. Це особливо корисно при використанні блокчейн-технологій, де часто необхідно точно визначити формат даних, які передаються через смарт-контракти.

```
interface ContractDetails {
    name: string;
    creator: string;
    address: string;
}

const contract: ContractDetails = {
    name: "MySmartContract",
    creator: "evdokim",
    address: "0xabcdef1234567890"
```

Дженерики: TypeScript підтримує використання дженериків, що дає можливість створювати функції чи класи, які працюють з різними типами даних, зберігаючи при цьому типову безпеку. Це надає можливість дозволяти та розробляти універсальні та повторно використовувані компоненти, що особливо вигідно при створенні смарт-контрактів для різноманітних токенів та активів.

```
function identity<T>(arg: T): T {
    return arg;
}

let output = identity<string>("Bonjoure");
```

Наявність цих можливостей значно полегшують побудову складних систем смарт-контрактів, роблячи їх більш адаптивними та здатними до розширення.

Розвинута екосистема: підтримка сучасних фреймворків (React, Node.js).

TypeScript має потужну спільноту та активно підтримує інтеграцію з численними популярними фреймворками і бібліотеками, такими як React, Angular, Vue.js, Node.js та іншими. Це відкриває доступ до широкого спектра інструментів для розробників, включаючи таке. За допомогою TypeScript ми

маємо відмінну сумісність з Node.js, це буде дозволяти створювати серверні додатки, які можуть взаємодіяти з блокчейн-мережами для запуску смарт-контрактів або збору та обробки даних із блокчейну.

React та інші фреймворки для фронтенду надають можливість підтримки у TypeScript з такими популярними фреймворками, як React та Angular, дозволяє розробникам створювати інтерфейси для децентралізованих додатків (dApps), що взаємодіють з блокчейнами та працюють із смарт-контрактами. React у TypeScript - це потужна комбінація, яка дозволяє розробникам створювати масштабовані та безпечні веб-додатки. Використання TypeScript з React надає ряд переваг, таких як статична типізація, що дозволяє виявляти помилки на етапі компіляції, а не під час виконання. Це особливо корисно в великих проектах, де складність коду може призводити до труднощів у відлагодженні.

При створенні компонентів у TypeScript можна визначати типи для пропсів і стану, що робить код більш безпечним. Наприклад, можна створити інтерфейс для пропсів і використовувати його в компоненті. Також, для класових компонентів можна типізувати стан, що дозволяє чітко визначити, які значення можуть бути у стані компонента. Використання хуків також можна типізувати, що дозволяє точно визначати типи значень, які вони повертають

Це дає змогу застосовувати TypeScript для всебічної розробки децентралізованих додатків, охоплюючи як фронтенд, так і бекенд, забезпечуючи безперебійну взаємодію всіх частин системи.

2 ОГЛЯД Й АНАЛІЗ ЛІТЕРАТУРНИХ,НАУКОВИХ ДЖЕРЕЛ

2.1 Огляд сучасних досліджень і джерел

У цьому розділі здійснено детальний огляд сучасних досліджень і літератури, присвячених розробці смарт-контрактів. Особливу увагу приділено аспектам, що мають значення для оцінки можливостей застосування TypeScript у цій сфері, зокрема його компіляції у Solidity. Наприклад модифікаторів Solidity це спеціальні функції, які змінюють поведінку інших функцій, дозволяючи додавати умови або логіку перед їх виконанням. Вони оголошуються за допомогою ключового слова modifier і можуть використовуватися для реалізації перевірок доступу, обмежень або виконання певних дій. Модифікатори застосовуються до функцій, що забезпечує організованість і безпеку коду. Наприклад, модифікатор onlyOwner може перевіряти, чи є викликач функції власником контракту, а модифікатор onlyWhenActive - чи активний контракт. Модифікатори можуть також приймати параметри, що робить їх більш гнучкими. Використання модифікаторів дозволяє створювати чистий код, зберігаючи логіку доступу та перевірок у централізованому місці, що підвищує безпеку і зрозумілість контрактів.

Основи принципи блокчейн та смарт-контрактів.

Перед аналізом використання TypeScript необхідно розглянути фундаментальні поняття та ключові принципи, які лежать в основі технологій блокчейну та смарт-контрактів, адже це створює основу для глибшого розуміння подальших досліджень. Смарт-контракти - це спеціалізовані програми, що розміщуються у блокчейні та виконуються автоматично при дотриманні певних умов. Завдяки їм можна реалізовувати угоди безпосередньо між сторонами без залучення посередників, що підвищує ефективність і прозорість процесів

Мови програмування для створення смарт-контрактів.

Для написання смарт-контрактів використовується кілька спеціалізованих мов програмування. Найпопулярнішою серед них є Solidity, яка розроблена спеціально для роботи на платформі Ethereum і є стандартом у цій сфері.) Solidity: Статично типізована мова програмування, створена спеціально для написання

смарт-контрактів, що працюють на Ethereum Virtual Machine (EVM). Вона має синтаксис, схожий на JavaScript і C++, що робить її зрозумілою для багатьох розробників. Серед переваг Solidity - велика база користувачів, розвинена екосистема та доступність широкого спектра інструментів для розробки. Водночас мова має і недоліки: висока ймовірність появи вразливостей, складнощі у створенні складних контрактів та обмежені можливості в порівнянні з іншими сучасними мовами. Vyper: Альтернатива Solidity, що пропонує простіший синтаксис та орієнтована на безпеку. Мова нагадує Python, завдяки чому є зрозумілою для розробників з досвідом роботи в цій екосистемі. Rust відома як системна мова програмування, що відзначається високою продуктивністю та безпекою. Move - мова програмування, спеціально розроблена компанією Facebook для блокчейн-платформи Diem (раніше відомої як Libra). Її ключовими особливостями є акцент на забезпеченні безпеки та можливість формальної верифікації, що дозволяє мінімізувати ризики помилок та вразливостей у смарт-контрактах. AssemblyScript - спрощена версія TypeScript, яка здатна компілювати код у формат WebAssembly (Wasm). Завдяки цій властивості AssemblyScript активно використовується для створення смарт-контрактів на блокчейн-платформах із підтримкою Wasm, таких як NEAR Protocol. Ця мова відкриває широкі можливості для розробників, які звикли працювати з TypeScript, пропонуючи їм зручний інструмент для написання контрактів у знайомому середовищі.

Інструменти для створення та розгортання смарт-контрактів.

Існує багато спеціалізованих інструментів, які значно спрощують процес написання, тестування й впровадження смарт-контрактів у блокчейн-мережі. Remix IDE - це веб-орієнтована інтегрована середовище розробки, створена для програмування на Solidity. Truffle Suite – це комплексний фреймворк, який забезпечує функціонал для створення, тестування та розгортання смарт-контрактів у мережі Ethereum.

Truffle Suite як комплексний фреймворк, який забезпечує функціонал для створення, тестування та розгортання смарт-контрактів у мережі Ethereum.

Hardhat як потужний фреймворк, призначений для розробки децентралізованих додатків (dApps) та смарт-контрактів на платформі Ethereum, з розширеними можливостями для налагодження і роботи з локальними блокчейнами

2.2 Література та її аналіз

Аналіз сучасної літератури щодо можливостей мови програмування TypeScript для розробки смарт-контрактів вказує на зростаючий інтерес до використання цієї мови в контексті децентралізованих технологій та блокчейн-інфраструктури. Однією з основних переваг TypeScript є його статична типізація, яка дозволяє виявляти помилки на етапі компіляції, що суттєво знижує ризик виникнення помилок у коді, які можуть мати серйозні наслідки в середовищі смарт-контрактів. Дослідження показують, що типізація допомагає розробникам створювати більш надійний і зрозумілий код, що є важливим аспектом для забезпечення безпеки смарт-контрактів.

Крім того, TypeScript має високу сумісність з уже існуючими JavaScript-бібліотеками, такими як Web3.js і Ethers.js, що дозволяє розробникам легко інтегрувати смарт-контракти у децентралізовані додатки (dApps). Це відкриває нові можливості для створення комплексних рішень у різних сферах, таких як фінансові послуги, управління цифровими активами, децентралізоване фінансування (DeFi) та системи голосування. Наприклад, використання TypeScript у проектах DeFi може суттєво підвищити надійність і безпеку фінансових транзакцій.

Сучасні дослідження також акцентують увагу на перевагах TypeScript у контексті сучасних фронтенд-фреймворків, таких як React, Angular та Vue.js. Ці фреймворки активно використовуються для створення інтуїтивно зрозумілих інтерфейсів для dApps, і TypeScript надає можливість розробникам використовувати потужні інструменти для перевірки типів, що дозволяє зменшити кількість помилок на етапі розробки.

Проте, література також вказує на деякі виклики, з якими стикаються розробники при використанні TypeScript для написання смарт-контрактів. По-

перше, TypeScript не є нативною мовою для блокчейн-розробки, як, наприклад, Solidity, що може призвести до обмежень у реалізації специфічних функцій, притаманних традиційним мовам програмування для смарт-контрактів. По-друге, існує потреба в адаптації існуючих інструментів і бібліотек для підтримки TypeScript, що може ускладнити процес розробки.

Дослідження також підкреслюють важливість безпеки у контексті розробки смарт-контрактів на TypeScript. Оскільки помилки в коді можуть призвести до вразливостей, які можуть бути використані зловмисниками, важливо проводити детальний аналіз безпеки та тестування смарт-контрактів, написаних на TypeScript. Це включає в себе використання статичного аналізу коду, тестування на вразливість та рецензії коду з боку експертів.

У підсумку, сучасні дослідження вказують на великий потенціал TypeScript для вдосконалення процесу розробки смарт-контрактів, підвищення надійності та безпеки коду, а також інтеграції з сучасними фронтенд-технологіями. Однак, для досягнення максимальних результатів необхідно продовжувати вивчення специфіки використання TypeScript у цій галузі, а також розробляти нові інструменти та методології, що допоможуть подолати існуючі виклики.

2.3 Новизна та актуальність

Хоча TypeScript вже широко використовується в розробці веб-додатків, його потенціал у контексті смарт-контрактів залишається недостатньо вивченим. Дослідження зосереджуються на можливостях цієї мови, що дозволяє виявити нові підходи до розробки смарт-контрактів, які можуть підвищити їхню надійність та безпеку. Аналіз переваг TypeScript у порівнянні з традиційними мовами програмування для смарт-контрактів, такими як Solidity, Vyper та Rust, дозволяє виявити нові аспекти, які можуть бути корисними для розробників. Це включає в себе статичну типізацію, покращену документацію та інтеграцію з існуючими JavaScript-бібліотеками. Дослідження можливостей інтеграції TypeScript з популярними фронтенд-фреймворками (React, Angular, Vue.js) для створення децентралізованих додатків (dApps) відкриває нові горизонти у

розробці інтерфейсів для смарт-контрактів. Вивчення аспектів безпеки, пов'язаних з використанням TypeScript для написання смарт-контрактів, є новим напрямком, що дозволяє оцінити, як статична типізація та інші особливості TypeScript можуть зменшити ймовірність виникнення вразливостей у контрактах.

Зростання популярності децентралізованих фінансів (DeFi). У світлі швидкого розвитку DeFi та блокчейн-технологій виникає необхідність у створенні надійних та безпечних смарт-контрактів. Дослідження нових мов програмування, таких як TypeScript, може суттєво підвищити якість коду в цій сфері. Популярність TypeScript серед веб-розробників створює запит на дослідження його можливостей у нових областях, таких як розробка смарт-контрактів, що може залучити нові таланти до цієї сфери.

Таким чином, вивчення можливостей TypeScript для розробки смарт-контрактів є надзвичайно важливим напрямком, що може значно вплинути на розвиток децентралізованих технологій і забезпечення їхньої безпеки.

2.4 Висновки по літературі

TypeScript пропонує статичну типізацію, що дозволяє виявляти помилки на етапі компіляції. Це значно підвищує надійність коду смарт-контрактів, зменшуючи ризик виникнення вразливостей, які можуть бути використані зловмисниками.

Використання TypeScript сприяє створенню більш зрозумілої та структурованої документації завдяки можливостям коментування та типізації, що полегшує роботу з кодом для інших розробників.

TypeScript є надбудовою над JavaScript, що забезпечує легку інтеграцію з існуючими бібліотеками, такими як Web3.js та Ethers.js. Це відкриває нові можливості для розробників, які вже мають досвід роботи з JavaScript. TypeScript активно підтримується популярними фронтенд-фреймворками, такими як React, Angular та Vue.js, що дозволяє створювати інтерактивні децентралізовані додатки (dApps) з високим рівнем продуктивності та безпеки.

Проведенний аналіз по реалізації.Література також містить аналіз різних підходів до реалізації смарт-контрактів на TypeScript, включаючи використання фреймворків, таких як Hardhat та Truffle, які спрощують процес розробки, тестування та деплою.

Проблеми та обмеження також виникають у сфері літературі. Тож незважаючи на численні переваги, існують певні виклики, такі як обмежена підтримка специфічних функцій блокчейн-програмування та необхідність адаптації до нових стандартів та практик у цій швидко змінюваній галузі.

Аналіз підходів для реалізації смарт-контрактів на мові TypeScript вимагає вивчення кількох ключових напрямків. По-перше, важливо розглянути основи TypeScript, включаючи його статичну типізацію, інтерфейси та класи, а також порівняти цю мову з традиційними мовами, такими як Solidity, Vyper та Rust. Далі слід звернути увагу на популярні фреймворки, такі як Hardhat і Truffle, які підтримують TypeScript, а також інструменти для тестування та деплою смарт-контрактів. Взаємодія з блокчейном через бібліотеки Web3.js та Ethers.js є ще одним важливим аспектом , що включає практичні приклади коду. Стосовно більш ретельнішого аналізу є можливість дізнатися використовуючи джерело [4].Безпека смарт-контрактів, написаних на TypeScript, також потребує уваги, з акцентом на вразливості та роль статичної типізації в їх уникненні. Крім того, корисно вивчити реальні кейс-стаді, які демонструють переваги TypeScript, а також поточні тренди та прогнози щодо його розвитку в контексті блокчейн-технологій і децентралізованих фінансів (DeFi). Завершуючи, методології дослідження, які включають порівняльний аналіз та опитування експертів, допоможуть оцінити як теоретичні, так і практичні аспекти використання TypeScript у розробці безпечних та надійних смарт-контрактів.

Таким чином, результати огляду літератури підтверджують актуальність та перспективність використання TypeScript для розробки смарт-контрактів, а також необхідність подальших досліджень у цій галузі для вдосконалення практик програмування та забезпечення безпеки децентралізованих додатків.

3 ПОСТАНОВКА ЗАДАЧІ

Наукова робота зосереджена на вивченні потенціалу мови програмування TypeScript у сфері розробки смарт-контрактів, з особливим акцентом на можливість компіляції в Solidity та вдосконалення супутніх інструментів для розробників. Актуальність цього дослідження обумовлена зростанням складності сучасних децентралізованих додатків (dApps) і контрактів, а також прагненням програмістів використовувати типобезпечні, знайомі та ефективні інструменти з екосистеми Web2. TypeScript розглядається як перспективний інструмент у блокчейн-розробці завдяки своїй здатності покращувати надійність коду, оптимізувати процес розробки та забезпечувати масштабованість проєктів.

Попри те, що TypeScript активно використовується для створення інфраструктури навколо смарт-контрактів на Solidity, зокрема для написання тестів, скриптів розгортання, конфігурацій і клієнтської логіки, ідея безпосередньої компіляції смарт-контрактів, написаних на TypeScript, у Solidity або байткод для EVM ще не має зрілого та загальновизнаного рішення. Це створює певний функціональний розрив, змушуючи розробників працювати з двома різними мовами одночасно, що ускладнює процес розробки, підвищує ймовірність помилок та збільшує навантаження.

Ця робота має на меті не лише оцінити сучасний стан інтеграції TypeScript у розробку смарт-контрактів, але й окреслити потенційні напрямки для подальших досліджень і розробок, які можуть підвищити доступність, безпеку та ефективність цього процесу. Особливу увагу приділено адаптації знайомих інструментів і підходів з екосистеми Web2 до специфіки Web3, що дозволить знизити поріг входу для нових розробників і уніфікувати стек технологій навколо TypeScript. Це відкриває можливості для створення нових рішень, таких як компілятори, проміжні представлення або фреймворки, які дозволять реалізовувати смарт-контракти за допомогою TypeScript або з мінімальними змінами контексту між мовами. Метою дослідження є оцінка ефективності та переваг використання TypeScript в порівнянні з традиційними методами програмування на Solidity, а також вивчення потенціалу для створення

інструментів та підходів, які б забезпечили безперешкодну інтеграцію цих технологій у процес розробки смарт-контрактів.

3.1 Огляд списку використаних технологій

У розробці смарт-контрактів з використанням TypeScript можна застосовувати кілька технологій та інструментів, які сприяють підвищенню ефективності та зручності процесу. По-перше, TypeScript є основною мовою програмування, яка забезпечує статичну типізацію, що дозволяє зменшити кількість помилок на етапі розробки та підвищує читабельність коду.

Solidity, в свою чергу, є основною мовою для написання смарт-контрактів на платформі Ethereum. TypeScript може використовуватися для написання тестів, скриптів для деплою та інших супутніх інструментів. Для цього розробники можуть використовувати середовища для розробки, такі як Hardhat, яке підтримує TypeScript і дозволяє писати, тестувати та деплоїти смарт-контракти, а також має плагіни для інтеграції з TypeScript. Іншим популярним середовищем є Truffle, яке також підтримує TypeScript через налаштування проекту і надає зручні інструменти для тестування та управління контрактами.

Node.js ідеально підходить для створення масштабованих мережевих додатків, таких як веб-сервери, реальні часи додатки (наприклад, чати або ігри) та API. Завдяки великій екосистемі модулів, доступних через npm (Node Package Manager), розробники можуть швидко інтегрувати різноманітні функціональні можливості у свої проекти.

Node.js також підтримує використання TypeScript, що дозволяє розробникам використовувати статичну типізацію та інші функції, що покращують якість коду. Це робить Node.js популярним вибором серед розробників, які прагнуть створювати сучасні, продуктивні та підтримувані веб-додатки.

Для покращення інтеграції TypeScript у процес розробки смарт-контрактів існує Hardhat TypeScript Plugin, який дозволяє писати скрипти та тести на TypeScript. Також варто згадати бібліотеку OpenZeppelin, яка надає стандартні смарт-контракти, що можуть бути використані разом з TypeScript для створення

безпечних і перевірених контрактів. Нарешті, фреймворк NestJS, призначений для побудови серверних додатків на TypeScript, може бути використаний для створення бекенду для децентралізованих додатків (dApps).

Бібліотека ts-morph - це непоганий варіант для роботи з TypeScript, яка надає зручний API для маніпуляцій з абстрактним синтаксичним деревом (AST) TypeScript. Вона дозволяє розробникам легко аналізувати, модифікувати та генерувати TypeScript код, що робить її корисною для створення інструментів, плагінів, трансформерів коду, а також для автоматизації рутинних завдань у проектах на TypeScript. Як аналог слід зазначити про вбудований TypeScript Compiler API, який надає TypeScript для роботи з AST. Хоча він менш зручний у використанні порівняно з ts-morph, він все ще є потужним інструментом для аналізу та трансформації TypeScript коду

Існує бібліотека child_process,дозволяє створювати нові процеси, які можуть виконувати команди або програми в системі. Це особливо корисно, коли потрібно виконати завдання, яке може зайняти певний час, або коли потрібно інтегруватися з іншими програмами.

Ці технології дозволяють розробникам максимально ефективно використовувати TypeScript у контексті розробки смарт-контрактів, знижуючи ризики помилок і підвищуючи продуктивність розробки.

4 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів акцентує увагу на інноваційних підходах, які можуть суттєво спростити процес створення децентралізованих додатків. TypeScript, завдяки своїй статичній типізації та інтеграції з сучасними інструментами розробки, відкриває нові горизонти для забезпечення надійності та безпеки коду.

У рамках аналізу були вивчені різноманітні фреймворки, такі як Hardhat та Truffle, які підтримують TypeScript, що дозволяє розробникам писати тестові сценарії та сценарії розгортання з використанням знайомої синтаксичної структури. Дослідження також звертає увагу на можливості використання бібліотек, таких як ethers.js та web3.js, для інтеграції TypeScript у взаємодію з блокчейн-мережами, що робить розробку більш інтуїтивною.

Крім того, важливим аспектом є розробка нових компіляторів та інструментів, які можуть перетворювати код, написаний на TypeScript, у формати, сумісні з Ethereum Virtual Machine (EVM). Це дозволить знизити навантаження на розробників, які наразі змушені працювати з кількома мовами програмування, зменшуючи ризик помилок і підвищуючи продуктивність.

Також дослідження підкреслює необхідність створення документації та навчальних матеріалів, щоб полегшити входження нових розробників у світ блокчейн-технологій, зокрема тих, хто вже знайомий з TypeScript. Впровадження TypeScript у розробку смарт-контрактів може стати важливим кроком до створення більш доступних і безпечних децентралізованих додатків, що відповідають сучасним вимогам ринку.

4.1 Основи та концептуальні принципи смарт-контрактів

Смарт-контракт - це самовиконувана програма, яка автоматично реалізує умови угоди на основі заздалегідь визначених параметрів у децентралізованих мережах, таких як блокчейн. Ці контракти дозволяють сторонам виконувати угоди і забезпечувати дотримання умов без необхідності залучення посередників або сторонніх організацій. Зазвичай смарт-контракт включає код, що визначає

умови угоди, а також чітко встановлені правила, які повинні бути виконані для активації або виконання контракту. Оскільки ці контракти зберігаються в блокчейні, вони набувають низки важливих характеристик. Автономність як виконання умов контракту здійснюється без залучення людини чи будь-яких сторонніх організацій. Як тільки умови виконуються, необхідні дії виконуються автоматично. Та прозорість усі дані щодо контрактів є відкритими для перевірки, що дає змогу перевірити правильність виконання умов усім учасникам мережі. Прозорість вся інформація, що стосується контрактів, є доступною для перевірки, що дає можливість кожному учаснику мережі перевірити коректність виконання умов. Прозорість вся інформація, що стосується контрактів, є доступною для перевірки, що дає можливість кожному учаснику мережі перевірити коректність виконання умов.

Розглядаючи контекст взаємодії у випадку TypeScript та Solidity, тоді мова програмування TypeScrip буде виступати важливим інструментом для розробників, оскільки вона дозволяє створювати інтерфейси та розробляти взаємодію з уже існуючими смарт-контрактами. Водночас, на відміну від Solidity, яка вже буде основною мовою для створення смарт-контрактів, TypeScript буде використовуватися переважно для створення та розробки децентралізованих додатків (dApps), що будуть взаємодіюти з блокчейном.

4.1.1 Основні принципи функціонування смарт-контрактів у блокчейн-мережах.

Під час проведення теоретичного дослідження слід зазначити ,що смарт-контракти є самовиконуваними угодами, які автоматично виконують умови, закладені в їх код, без необхідності втручання третіх осіб. Вони функціонують на децентралізованих платформах, таких як Ethereum, що забезпечує прозорість та безпеку угод, оскільки не залежать від центрального управління. Незмінність коду смарт-контракту після його розгортання в блокчейн гарантує, що умови контракту залишаться незмінними і не підлягатимуть маніпуляціям.

Смарт-контракти пишуться на спеціалізованих мовах програмування, таких як Solidity. Проте TypeScript може бути використаний для створення фронтенд-інтерфейсів або для взаємодії з контрактами через бібліотеки, що полегшує інтеграцію з існуючими системами. Завдяки своїй статичній типізації та підтримці сучасних бібліотек, таких як ethers.js і web3.js, TypeScript дозволяє розробникам ефективно реалізовувати логіку для взаємодії з цими контрактами.

Оскільки смарт-контракти часто обробляють фінансові транзакції, їх безпека є критично важливою. Використання TypeScript може допомогти зменшити кількість помилок у коді завдяки статичній типізації, що, в свою чергу, знижує ризики уразливостей. Розробка смарт-контрактів також потребує ретельного тестування та аудиту. TypeScript, разом з такими фреймворками, як Hardhat, надає інструменти для ефективного тестування контрактів, що дозволяє виявляти та виправляти помилки до їх розгортання.

Отже, дослідження можливостей TypeScript для розробки смарт-контрактів відкриває нові горизонти для розробників, які прагнуть створювати безпечні, масштабовані та ефективні рішення у блокчейн-екосистемі. Використання TypeScript може значно спростити інтеграцію з існуючими смарт-контрактами, покращуючи процес розробки та забезпечуючи високий рівень якості коду.

4.1.2 Вивчення прогалини у Solidity та TypeScript

Завдяки теоретичному дослідження можна дізнатися про використання TypeScript, яка мови програмування набуває все більшої популярності для розробки додатків, які забезпечують інтерактивність з різними блокчейн-мережами. Завдяки своїй статичній типізації, TypeScript дозволяє розробникам ефективно взаємодіяти з уже існуючими смарт-контрактами, використовуючи спеціалізовані бібліотеки та фреймворки, такі як ethers.js і web3.js, які надають потужні інструменти для комунікації з блокчейн-системами.

Цей підхід не лише сприяє створенню типобезпечних і масштабованих додатків, але й значно підвищує надійність систем, зменшуючи ймовірність помилок у коді. Крім того, TypeScript дозволяє розробникам легко організовувати

код, завдяки чому інтеграція з існуючими смарт-контрактами стає більш зручною та швидкою.

Важливо зазначити, що завдяки активній підтримці спільноти та розвитку нових фреймворків, таких як Hardhat, розробка на TypeScript стає більш доступною для новачків у світі блокчейну. Це відкриває нові можливості для інновацій, адже розробники можуть швидше реалізовувати ідеї, тестувати їх у безпечному середовищі, а також адаптувати свої проекти до змінюваних умов ринку. Таким чином, використання TypeScript у блокчейн-розробці не лише підвищує ефективність, але й сприяє створенню більш безпечних та надійних дистрибутивних систем.

Проведення теоритичного дослідження над потенціалом мови програмування TypeScript у контексті розробки смарт-контрактів відкриває нові горизонти для програмістів, які прагнуть інтегрувати сучасні технології блокчейну у свої проекти. Один з основних аспектів, які роблять TypeScript привабливим для цієї мети, - це статична типізація, що дозволяє виявляти помилки на ранніх етапах розробки. Це суттєво знижує ризики, пов'язані з помилками у коді, особливо в контексті фінансових транзакцій, де навіть незначна помилка може призвести до серйозних наслідків.

Крім того, TypeScript легко інтегрується з вже відомими інструментами, такими як Hardhat і Truffle, що забезпечує зручне середовище для тестування та розгортання смарт-контрактів. Хоча наразі існує обмежена кількість рішень для компіляції TypeScript у байт-код, що використовує Ethereum Virtual Machine

4.1.3 Виклики та різниця в семантиці

TypeScript і Solidity - це дві різні мови програмування, які призначені для різних цілей, і вони мають свої унікальні виклики та семантичні особливості. Одним з основних викликів для Solidity є безпека, оскільки смарт-контракти обробляють фінансові транзакції. Розробники повинні бути дуже уважними до можливих уразливостей, таких як переповнення буфера та атаки повторного використання, оскільки кожен рядок коду може мати серйозні наслідки. У той же

час, TypeScript також має свої виклики безпеки, особливо при взаємодії з API та зовнішніми сервісами. Хоча статична типізація в TypeScript допомагає зменшити кількість помилок, вона не усуває їх повністю.

Середовище виконання також відрізняється для цих двох мов. Solidity виконується на віртуальній машині Ethereum (EVM), що накладає певні обмеження, зокрема обмеження на газ, які можуть впливати на проектування смарт-контрактів. У свою чергу, TypeScript виконується в середовищі браузера або Node.js, що надає розробникам доступ до широких можливостей JavaScript-екосистеми, але вимагає врахування різних платформ і браузерів. Дебагінг смарт-контрактів на Solidity може бути складним через децентралізовану природу блокчейн-технологій, тоді як дебагінг у TypeScript зазвичай простіший завдяки потужним інструментам розробки.

Що стосується семантики, Solidity використовує динамічну типізацію з обмеженим набором типів даних, специфічних для блокчейн-технологій, таких як address і uint. Розробники повинні бути обережними при управлінні типами. У той же час TypeScript підтримує статичну типізацію, що дозволяє визначати типи змінних і функцій, сприяючи кращій перевірці коду на етапі компіляції. Ще однією важливою відмінністю є концепція контрактів у Solidity, яка дозволяє створювати децентралізовані додатки з власним станом і функціями. На відміну від цього, TypeScript більше зосереджений на об'єктно-орієнтованих принципах, не маючи концепції «контрактів» у такому сенсі.

В цілому, виклики та семантичні особливості TypeScript і Solidity відображають їх різні призначення та контексти використання. Розуміння цих відмінностей є важливим для розробників, які прагнуть створити безпечні та ефективні рішення в світі блокчейн-технологій.

4.1.4 Рішення та роль проміжного представлення (IR)

Проміжні представлення (IR) відіграють важливу роль у процесі компіляції та виконання коду, зокрема в контексті мов програмування, таких як Solidity і TypeScript. IR є абстрактним представленням вихідного коду, яке дозволяє

компіляторам та іншим інструментам виконувати оптимізації та трансформації, не прив'язуючись до конкретної мови або архітектури. Це дає змогу розробникам створювати більш ефективні та безпечні програми, оскільки IR може бути використане для аналізу та оптимізації коду на різних етапах компіляції.

У контексті Solidity, IR може допомогти виявити потенційні вразливості в смарт-контрактах ще до їх розгортання на блокчейні. Наприклад, інструменти аналізу можуть перетворювати код Solidity у IR для виявлення проблем, таких як переповнення чисел або неправильне управління доступом. Це дозволяє розробникам виправити помилки на ранніх етапах, зменшуючи ризики, пов'язані з безпекою контрактів. Використання IR також може полегшити процес тестування, оскільки дозволяє розробникам перевіряти логіку смарт-контрактів без необхідності їх безпосереднього виконання на блокчейні.

У випадку TypeScript, IR може бути корисним для оптимізації коду перед його перетворенням на JavaScript. Це дозволяє компіляторам виконувати різноманітні оптимізації, такі як видалення непотрібного коду або поліпшення продуктивності, що особливо важливо для веб-додатків, де швидкість виконання є критично важливою. IR також може використовуватись для статичного аналізу, що допомагає виявити помилки та покращити якість коду на етапі розробки. Завдяки цьому розробники можуть зосередитися на бізнес-логіці, залишаючи технічні деталі компіляторам та інструментам аналізу.

Отже, проміжні представлення є ключовими компонентами в екосистемі розробки програмного забезпечення, оскільки вони забезпечують гнучкість, безпеку та продуктивність. Вони дозволяють розробникам зосередитися на високорівневих аспектах програмування, при цьому забезпечуючи ефективність та надійність кінцевого коду. Використання IR у Solidity та TypeScript підкреслює важливість цього підходу для створення якісних рішень у світі сучасних технологій.

4.2 Проведення паралельної компіляції у сфері TypeScript та Solidity

Паралельна компіляція є важливим аспектом у розробці програмного забезпечення, оскільки вона дозволяє зменшити час компіляції, підвищити продуктивність та ефективність розробки. У сфері TypeScript та Solidity, паралельна компіляція може бути реалізована через використання різних інструментів та технологій, що дозволяє одночасно обробляти кілька файлів коду. Це особливо корисно у великих проектах, де кількість файлів і модулів може бути значною, і де час компіляції може суттєво вплинути на загальний процес розробки.

У випадку TypeScript, паралельна компіляція може бути досягнута за допомогою таких інструментів, як **tsc** (TypeScript Compiler) у поєднанні з системами зборки, такими як Webpack або Parcel. Ці інструменти дозволяють розробникам налаштовувати процес компіляції таким чином, щоб одночасно обробляти кілька модулів, що прискорює збірку проекту. Крім того, TypeScript підтримує функціонал, що дозволяє компілювати лише змінені файли, що ще більше зменшує час, необхідний для компіляції.

З іншого боку, у Solidity паралельна компіляція може бути реалізована за допомогою фреймворків, таких як Hardhat або Truffle. Ці інструменти дозволяють компілювати кілька смарт-контрактів одночасно, що значно підвищує ефективність розробки, особливо в умовах, коли проект містить безліч контрактів. Паралельна компіляція в Solidity також може включати оптимізації, які зменшують загальний розмір байт-коду, що в свою чергу знижує витрати на газ при розгортанні контрактів на блокчейні.Як приклад розглянемо таблицю паралельна компіляції в TypeScript та Solidity – рисунок 2. Таким чином, паралельна компіляція у TypeScript та Solidity демонструє різні підходи до оптимізації процесу компіляції в залежності від специфіки кожної мови. Обидві мови надають розробникам можливості для підвищення продуктивності, але реалізація цих можливостей може суттєво відрізнятися через різні середовища виконання та специфіку розробки.

Характеристика	TypeScript	Solidity
Інструменти	`tsc`, Webpack, Parcel	Hardhat, Truffle
Тип компіляції	Одночасна компіляція кількох модулів	Одночасна компіляція кількох смарт-контрактів
Оптимізації	Компіляція лише змінених файлів	Оптимізація байт-коду для зменшення витрат на газ
Загальний вплив на продуктивність	Зменшення часу збірки проекту	Зменшення часу розгортання смарт-контрактів
Середовище виконання	Браузер, Node.js	Ethereum Virtual Machine (EVM)
Складність налагодження	Менш складна; наявність потужних інструментів	Більш складна; обмежена доступність інструментів

Рисунок 2 – Таблиця Паралельна компіляції в TypeScript та Solidity(рисунок виконаний самостійно)

Отже, хоча паралельна компіляція може підвищити ефективність розробки в обох мовах, важливо враховувати їхні особливості та контекст використання. Рекомендовано продовжити дослідження в цій області, щоб знайти оптимальні рішення для інтеграції TypeScript у процеси розробки смарт-контрактів на Solidity. TypeScript, завдяки своїй гнучкій системі типізації та можливостям для динамічного програмування, дозволяє розробникам створювати складні веб-додатки з високим рівнем абстракції. Процес компіляції в TypeScript, як правило, швидший через можливість компіляції в JavaScript, що виконується у різних середовищах.

4.3 Масштабованість

Масштабованість у контексті дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів є одним із ключових аспектів, оскільки вона впливає на здатність системи адаптуватися до зростаючих вимог та складності проектів. TypeScript, будучи надмножиною JavaScript, надає розробникам статичну типізацію, що дозволяє виявляти помилки на етапі

компіляції, а не під час виконання. Це особливо важливо в контексті смарт-контрактів, де навіть незначна помилка може призвести до серйозних фінансових втрат або вразливостей у безпеці.

Однією з основних переваг TypeScript є його можливість підтримувати об'єктно-орієнтоване програмування, що дозволяє розробникам створювати модульні та повторно використовувані компоненти. Це означає, що смарт-контракти можуть бути розроблені як окремі модулі, які легко інтегруються один з одним, що значно спрощує управління великою кодовою базою. Наприклад, розробники можуть створювати бібліотеки контрактів, які реалізують загальні функціональні можливості, що дозволяє уникати дублювання коду і знижує ймовірність помилок.

Крім того, TypeScript інтегрується з популярними фреймворками для розробки смарт-контрактів, такими як Hardhat і Truffle. Ці інструменти забезпечують потужні можливості для тестування, налагодження та автоматизації процесів розробки, що сприяє підвищенню ефективності роботи команди. Наприклад, Hardhat надає можливість легко налаштовувати середовище для тестування, а також підтримує плагіни, які можуть розширювати функціональність проекту, що, в свою чергу, сприяє масштабованості.

Додатково, TypeScript дозволяє використовувати сучасні підходи до розробки, такі як CI/CD (безперервна інтеграція та безперервне постачання), що допомагає автоматизувати процеси тестування та розгортання смарт-контрактів. Це особливо важливо в умовах швидко змінюваного ринку, де нові функції та оновлення можуть бути впроваджені з мінімальними перервами в роботі

У підсумку, TypeScript забезпечує потужні інструменти і можливості для розробки масштабованих смарт-контрактів, що дозволяє розробникам створювати якісний, підтримуваний і безпечний код, здатний адаптуватися до зростаючих вимог бізнесу та технологічних змін.

4.4 Мета теоретичного дослідження та підсумки

Мета теоретичного дослідження полягає у всебічному аналізі використання TypeScript для розробки смарт-контрактів, з акцентом на аспекти масштабованості, безпеки, підтримуваності та ефективності коду. Основна мета - виявити, як специфікації TypeScript можуть вплинути на процес розробки децентралізованих додатків (dApps) і смарт-контрактів, порівнюючи їх з традиційними мовами програмування, такими як Solidity. Дослідження також намагається визначити, як TypeScript може допомогти в управлінні складністю проектів, що зростають, та в адаптації до змінюваних бізнес-вимог. Важливими аспектами є аналіз статичної типізації, об'єктно-орієнтованого програмування, а також інтеграція з фреймворками для тестування та автоматизації.

Підсумки дослідження свідчать про те, що TypeScript має ряд переваг, які роблять його привабливим для розробників смарт-контрактів. По-перше, статична типізація дозволяє виявляти помилки на етапі компіляції, що значно зменшує ймовірність появи вразливостей у коді смарт-контрактів. Це особливо важливо, оскільки смарт-контракти часто управляють значними фінансовими активами, і будь-яка помилка може призвести до серйозних наслідків.

По-друге, TypeScript підтримує об'єктно-орієнтовані принципи, що дозволяє розробникам створювати модульні, повторно використовувані компоненти. Це не тільки спрощує управління великими кодовими базами, але й полегшує командну роботу, оскільки різні розробники можуть працювати над різними модулями без ризику конфлікту. Також, завдяки своїй структурованій природі, TypeScript дозволяє легше впроваджувати нові функціональності без ризику зламати існуючий код.

Теоретичне дослідження також вказує на важливість масштабованості в контексті розробки смарт-контрактів. Системи, які не можуть адаптуватися до зростаючих вимог, ризикують стати застарілими або неефективними. TypeScript, завдяки своїй гнучкості та можливості створення розширюваних архітектур, дозволяє розробникам створювати рішення, які можуть легко адаптуватися до нових умов. Це особливо актуально в умовах швидко змінюваного світу

технологій, де підприємства постійно шукають способи оптимізації своїх процесів і підвищення продуктивності.

Таким чином, підсумовуючи результати теоретичного дослідження, можна стверджувати, що TypeScript є потужним інструментом для розробників смарт-контрактів, що забезпечує високий рівень якості коду, безпеки та масштабованості. Його можливості сприяють створенню надійних, підтримуваних і адаптивних рішень, що робить TypeScript перспективним вибором для розробки у сфері блокчейн-технологій.

4.5 Компіляція TypeScript в Solidity використувуючи ts-morph.

Загалом ts-morph - це потужна бібліотека, яка дозволяє маніпулювати кодом TypeScript за допомогою абстрактного синтаксичного дерева (AST). Використовуючи ts-morph, розробники можуть аналізувати, модифікувати та генерувати код TypeScript, що робить його корисним інструментом для трансформації коду з TypeScript в Solidity. У контексті компіляції, ts-morph може бути використаний для автоматизації процесу перетворення логіки, написаної в TypeScript, у відповідні структури та синтаксис Solidity, зберігаючи при цьому типи та бізнес-логіку. Це дозволяє розробникам, які вже знайомі з TypeScript, легко переходити до написання смарт-контрактів, зменшуючи час на навчання новій мові. Зокрема, завдяки можливостям ts-morph, можна створити інструменти, які автоматично генерують Solidity-код з TypeScript, що робить процес розробки більш ефективним і зменшує ймовірність помилок, пов'язаних з ручним переписуванням.

Теоретичний процес компіляції передбачає наступний ланцюг дій: спочатку ви розробляєте смарт-контракт на TypeScript, дотримуючись певних обмежень, оскільки не всі конструкції цієї мови можуть бути безпосередньо переведені на Solidity. Потім, за допомогою скрипта на Node.js у поєднанні з бібліотекою ts-morph, відбувається завантаження та парсинг TypeScript-файлів у абстрактне синтаксичне дерево (AST). Далі виконується семантичний аналіз і трансляція - це найскладніший етап, оскільки потрібно узгодити відмінності між TypeScript і

Solidity/EVM. Після цього генерується синтаксично коректний Solidity-код у вигляді .sol-файлів. На наступному етапі компілятор Solidity (solc) перетворює ці файли у байткод для Ethereum Virtual Machine (EVM), який є фінальним результатом, готовим до розгортання. Весь цей процес можна наочно представити у вигляді діаграми на рисунку – 3.

Крок	Опис
1. Початок	Написання смарт-контракту на TypeScript.
2. Обмеження TypeScript	Визначення обмежень TypeScript, оскільки не всі конструкції мають відповідники в Solidity.
3. Завантаження та парсинг	Використання Node.js та ts-morph для завантаження TypeScript-файлів та парсингу їх у абстрактне синтаксичне дерево (AST).
4. Семантичний аналіз	Проведення семантичного аналізу AST для перевірки типів та логіки коду.
5. Трансляція	Трансляція TypeScript-коду в структури Solidity з урахуванням специфіки мови.
6. Генерація Solidity-коду	Генерація синтаксично правильного Solidity-коду (.sol файли) з AST.
7. Компіляція в байткод	Використання компілятора Solidity (solc) для компіляції згенерованого Solidity-коду в байткод для Ethereum Virtual Machine (EVM).
8. Фінальний результат	Отримання байткоду, готового до розгортання на блокчейні.

Рисунок 3 – Процес компіляції використовуючи ts-morph (рисунок виконаний самостійно)

Solidity та TypeScript є мовами програмування, які мають різні призначення і підходи до компіляції. Solidity, спеціально розроблена для написання смарт-контрактів на платформі Ethereum, компілюється в байт-код, що виконується в Ethereum Virtual Machine (EVM). Цей процес включає перевірку синтаксису та семантики, що дозволяє виявляти помилки на ранніх етапах, але також вимагає

високої точності, оскільки навіть незначні помилки можуть призвести до серйозних наслідків під час виконання контракту.

На відміну від цього, TypeScript є надмножиною JavaScript, яка додає статичну типізацію та об'єктно-орієнтовані можливості. Код на TypeScript компілюється в чистий JavaScript, що робить його універсальним для виконання в різних середовищах. Хоча TypeScript також забезпечує статичну типізацію, що допомагає виявляти помилки до виконання, його компіляція в JavaScript може призвести до того, що деякі помилки залишаються непоміченими до моменту виконання.

Отже, основна різниця в компіляції між Solidity та TypeScript полягає в їхньому призначенні, механізмах перевірки типів і підходах до обробки помилок. Solidity вимагає більшої точності та обережності через критичність помилок у смарт-контрактах, тоді як TypeScript пропонує гнучкіший підхід до розробки з можливістю виявлення помилок на етапі компіляції. Обидві мови мають свої переваги, але вибір між ними залежить від специфічних потреб проекту.

5 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ

5.1 Проведення інтеграції під час розробки смарт-контрактів

Практична частина дослідження зосереджувалася на вивченні реальних прикладів використання та ефективних інструментів, які полегшують інтеграцію TypeScript у процес створення смарт-контрактів, з особливим акцентом на екосистему Ethereum Virtual Machine (EVM). Зважаючи на те, що наразі не існує стабільних і повноцінних рішень для безпосередньої компіляції TypeScript у Solidity-код, ключова увага була приділена засобам взаємодії між цими мовами, таким як ethers.js, TypeChain про яке ретельніше можна дізнатися опираючись на джерело [5], а також методам синхронної компіляції контрактів і TypeScript-коду.

У дослідженні також розглядалися підходи до організації безпечної та ефективної розробки Web3-застосунків, де TypeScript використовується як мова для написання логіки клієнта або серверної частини, яка взаємодіє з децентралізованою інфраструктурою. Середовище розробки на базі Hardhat, разом із сучасними практиками DevOps (наприклад, CI/CD для смарт-контрактів), продемонструвало високу ефективність у забезпеченні стабільної інтеграції та масштабування таких систем.

5.1.1 TypeScript та взає'модія з Web3

TypeScript застосовується разом із Node.js. Крім того, він активно використовується для написання скриптів розгортання та тестування смарт-контрактів, що робить його ключовим інструментом у всіх етапах розробки Web3-проєктів. TypeScript, як типобезпечне розширення JavaScript, відіграє дедалі важливішу роль у розробці Web3-застосунків, незважаючи на те, що сама мова не призначена для безпосереднього створення смарт-контрактів. Основним предметом дослідження є можливості застосування TypeScript у процесі реалізації смарт-контрактів, зокрема через створення ефективного, безпечного та типобезпечного програмного середовища для взаємодії з Ethereum Virtual Machine (EVM). Хоча смарт-контракти продовжують розроблятись переважно на Solidity або Vyper, роль TypeScript стає критично важливою на етапі побудови

інтерфейсів, логіки користувача і бекенд-частин, які взаємодіють з цими контрактами. TypeScript забезпечує зручність розробки за рахунок жорсткої типізації, що значно знижує ймовірність помилок при обробці даних, виклику функцій контрактів та обробці відповідей від блокчейну.

Ключовими інструментами, які дозволяють ефективно інтегрувати TypeScript у Web3-розробку, є ethers.js для підключення до Ethereum-мережі та здійснення транзакцій, а також TypeChain, який генерує типобезпечні обгортки на основі ABI контрактів. Це дає змогу писати код, у якому методи контракту мають чіткі сигнатури, а можливі помилки виявляються ще на етапі компіляції, а не під час виконання.

Особливу увагу під час дослідження було приділено таким інструментам, як Hardhat - середовищу розробки, яке підтримує написання скриптів розгортання, тестування та налагодження контрактів на TypeScript. Поєднання Hardhat із TypeScript дозволяє розробникам створювати повноцінні блокчейн-проєкти з високим ступенем автоматизації та надійності. Крім того, інтеграція таких інструментів, як The Graph, також здійснюється через TypeScript, що забезпечує гнучке індексування даних з блокчейну та створення GraphQL API.

Незважаючи на відсутність нативної підтримки компіляції TypeScript у байткод EVM, паралельне використання TypeScript у ролі логіки взаємодії з контрактами виявилось найбільш ефективним та адаптивним підходом у сучасній практиці Web3. Цей підхід також відкриває перспективи майбутніх експериментів із доменно-специфічними мовами або метакомпіляторами, які можуть забезпечити глибшу інтеграцію мов високого рівня з блокчейн-платформами.

Таким чином, TypeScript не лише спрощує життя Web3-розробникам, а й стає мостом між традиційною веброзробкою та децентралізованими технологіями. Його використання у поєднанні з інструментами на кшталт Ethers.js, TypeChain і Hardhat дозволяє створювати надійні, масштабовані і типобезпечні рішення, готові до роботи у блокчейн-середовищі.

У ході практичного дослідження стало очевидно, що навіть за наявності сучасного інструментарію, розробники Web3-застосунків стикаються з цілою

низкою викликів, пов'язаних із використанням одночасно двох мов - Solidity для створення смарт-контрактів та TypeScript для написання логіки децентралізованого застосунку (dApp). У реальних проєктах часто виникають труднощі на межі цих двох екосистем - це, зокрема, ручна перевірка типів, потреба у синхронізації ABI, а також помилки, пов'язані з невідповідністю між типами Solidity , стосовно якого слід використовувати гайд щодо використання з джерела [6] та типами в TypeScript. Такий розрив між середовищами ускладнює розробку і вимагає від команди додаткових зусиль для підтримки узгодженості коду.

Водночас результати дослідження підтвердили, що TypeScript відіграє фундаментальну роль у сучасній інфраструктурі Web3, не як альтернатива мові Solidity, а як надійний інструмент для інтеграції та обслуговування взаємодії з контрактами. Його сильна типізація, інтеграція з такими інструментами, як Ethers.js і TypeChain, а також підтримка автоматизації (наприклад, у Hardhat) забезпечують високу якість розробки, зниження кількості помилок і пришвидшення циклу створення dApp.У сучасній сфері розробки смарт контрактів Typechain відіграє важливу роль , опираючись на оцінку з джерела [7].

Попри те що для інших блокчейн-платформ вже створено окремі проєкти з компіляції TypeScript-подібних мов у байткод відповідних віртуальних машин, для EVM подібна ідея поки залишається концептуальною. А для якісної роботи слід зазначати що потрібно використовувати приклади ,яке гарно наведені у джерелі [8]. Але наразі відсутні продакшен-рішення, які б дозволяли напряму компілювати TypeScript у смарт-контрактний байткод Ethereum. Це означає, що в реальних умовах найбільш практичним підходом залишається паралельна розробка з чітким розділенням обов'язків між Solidity та TypeScript, де останній забезпечує надійність логіки взаємодії та комфорт розробника.

У ході практичного дослідження було встановлено, що хоча безпосередня компіляція TypeScript у байткод Ethereum Virtual Machine (EVM) наразі залишається недоступною, існують успішні приклади реалізації цього підходу на інших блокчейн-платформах, які побудовані на альтернативних віртуальних

машинах. Такі проєкти доводять, що використання TypeScript-подібного синтаксису для створення смарт-контрактів є технічно здійсненним, хоча й за межами EVM-сумісного середовища, в якості іншого фактору можна також продитивсь приклади інших моментів зі смарт-контрактами у джерелі [9].

Зокрема, у середовищі Internet Computer Protocol активно використовується проєкт Azle, який дозволяє писати смарт-контракти на TypeScript і компілювати їх у WebAssembly (WASM). Подібний підхід реалізовано у Casper і NEAR, де застосовується AssemblyScript - підмножина TypeScript, яка також транслюється у WASM. У блокчейні Cardano існує власне рішення - Plu-ts, що дозволяє розробляти смарт-контракти у вигляді DSL (domain-specific language), який зовні нагадує TypeScript і транслюється у внутрішній формат Plutus Core.

Ці приклади демонструють, що використання TypeScript не обмежується лише побудовою логіки взаємодії з контрактами, а може слугувати і як повноцінна мова для написання контрактів - але лише у відповідному технологічному середовищі. Для EVM така трансляція все ще залишається у сфері експериментальних ідей, без готових до широкого використання реалізацій. Це, однак, не знижує значущості TypeScript у Web3-розробці: він залишається ключовим елементом інфраструктури, забезпечуючи типобезпечну, ефективну та гнучку взаємодію з контрактами на Solidity. Проведення подібних у сфері методів у інформаційному джерелі [10] може навести на правильну думку стосовно проведення майбутніх досліджень у сфері покращення смарт-контрактів.

5.1.2 Компіляції Solidity

У межах дослідження, присвяченого вивченню потенціалу мови програмування TypeScript у контексті створення смарт-контрактів, було реалізовано один із експериментальних підходів до трансляції TypeScript-коду у формат, сумісний із Solidity. Зокрема, представлено приклад програмної моделі, яка ілюструє можливість автоматичного перетворення класів, написаних у стилі TypeScript, у вихідний код смарт-контрактів на мові Solidity. Повна реалізація цього підходу наведена в Додатку А.

Ключовим елементом даної моделі є клас ClassParser, розроблений на TypeScript. Його завдання полягає в аналізі структури класів TypeScript і генеруванні на основі цієї структури відповідного Solidity-коду. У класі передбачено поле options, що відповідає за параметри компіляції, серед яких, наприклад, форматування або розмір відступів. Конструктор приймає на вхід об’єкт типу ClassDeclaration, який репрезентує оголошення класу, а також необов’язковий об'єкт параметрів, що дозволяє гнучко змінювати налаштування компілятора.

```
constructor(
  private readonly cls: ClassDeclaration,
  options?: Partial<ClassParserOptions>,
) {
  this.options = { ...defaultOptions, ...options };
}
```

Основна логіка трансформації реалізована в методі compile. Цей метод поетапно будує компоненти майбутнього смарт-контракту, зокрема: назву контракту (getContractName), список змінних (getProperties), конструктор (getConstructor), а також функції (getMethods). Отримані фрагменти об’єднуються в єдиний текстовий шаблон, що формує повну структуру контракту на мові Solidity.

Ця реалізація демонструє один із можливих шляхів досягнення глибшої інтеграції TypeScript у процес створення смарт-контрактів. Хоча підхід залишається експериментальним, він підтверджує, що TypeScript може виступати не лише як інструмент взаємодії з контрактами, а й як основа для генерації самих контрактів, у межах суворо визначених синтаксичних правил. Це відкриває перспективу для подальших досліджень, спрямованих на створення повноцінних трансляторів або DSL, орієнтованих на Web3-розробку з використанням мов високого рівня.

```
public generateFunction(methodName) {
    const parameters = this.getMethodParameters(methodName)
        .map(param => this.formatParameter(param))
        .join(", ");
```

```
        const returnType = this.getReturnType(methodName);
        const body = this.getMethodBody(methodName);

        return `function ${methodName}(${parameters}) public returns
    (${returnType}) {\n${body}\n}`;
    }

    private formatParameter(parameter) {
        const { name, type } = parameter;
        return `${type} ${name}`;
    }
```

Метод generateFunction() призначений для автоматичного генерування рядка, що представляє функцію на мові Solidity. Він спрощує процес створення функцій у смарт-контрактах, що особливо корисно для розробників, які працюють з цими технологіями. Метод починає свою роботу з отримання параметрів для заданого методу, використовуючи функцію getMethodParameters(methodName). Ця функція повертає масив параметрів, кожен з яких містить інформацію про ім'я та тип. Параметри форматуються за допомогою допоміжного методу formatParameter(), що забезпечує правильний синтаксис у вигляді тип ім'я.

Далі, метод отримує тип повернення функції через getReturnType(methodName), що дозволяє визначити, який тип даних буде повернуто після виконання функції. Якщо функція не повертає значення, це також враховується. Після цього викликається getMethodBody(methodName), щоб отримати реалізацію методу у вигляді рядка. Це тіло методу може містити будь-яку логіку, необхідну для виконання функції.

На завершення, всі отримані елементи (назва методу, параметри, тип повернення та тіло) об'єднуються в один рядок, що представляє повну функцію на Solidity. Наприклад, для методу transfer, який приймає адресу отримувача та суму, результат може виглядати як function transfer(address recipient, uint amount) public returns (bool) {...}. Таким чином, метод generateFunction() не лише автоматизує створення функцій, але й забезпечує дотримання синтаксису Solidity, що зменшує ймовірність помилок під час компіляції. Це робить його важливим інструментом для розробників, які прагнуть оптимізувати свій робочий процес у створенні смарт-контрактів.

```
  } private getConstructor() {
    const constructorName = this.cls.getName();
    const parameters =
this.cls.getConstructorParameters().map((param) => {
        const paramName = param.getName();
        const paramType = param.getType().getText();
        return `${paramType} ${paramName}`;
    }).join(', ');

    return `constructor(${parameters}) {}`;
}
```

Метод getConstructor() генерує рядок, що представляє конструктор смарт-контракту в Solidity. Спочатку він отримує ім'я класу, яке буде використовуватися в якості імені конструктора, через метод getName(). Далі метод викликає getConstructorParameters(), щоб отримати масив параметрів конструктора. Для кожного параметра, за допомогою методу map(), він витягує ім'я параметра через getName() і тип через getType().getText().Ці дані об'єднуються в рядок формату тип ім'я, а потім всі параметри з'єднуються через кому, утворюючи частину конструктора. Врешті-решт, метод повертає рядок у форматі constructor(параметри) {}, що є стандартним синтаксисом для визначення конструктора в Solidity.

Цей метод автоматизує створення конструктора, що дозволяє розробникам легко визначати початкові параметри для смарт-контракту, забезпечуючи правильність синтаксису та зменшуючи ймовірність помилок під час написання коду.

```
private getMethodBody(statement: Statement) {
    const statementText = statement.getText();
    return `${statementText.replace("this.", "")}`;
  }

  private getDecoratorNames(decorators: Decorator[]) {
    return decorators.map((decorator) => decorator.getName()).join("
");
  }

  private getMethodParameters(param: ParameterDeclaration) {
    const paramName = param.getName();
    const paramType = param.getType().getText();
    return `${paramType} ${paramName}`;
```

```
}
```

Як ілюструє рисунок 4, процес починається з парсингу TypeScript-файлу, далі відбувається пошук усіх класів і побудова абстрактного синтаксичного дерева за допомогою бібліотеки ts-morph. Після цього вузли дерева аналізуються й трансформуються у відповідні конструкції Solidity, з яких і формується фінальний код смарт-контракту.

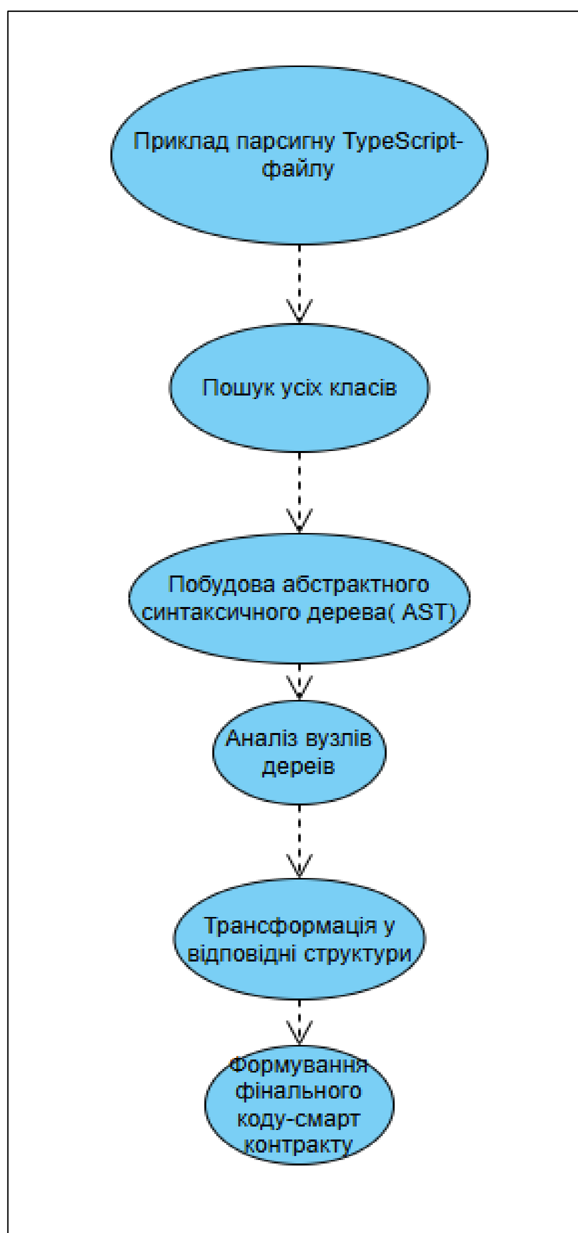

Рисунок 4 – Приклад діаграми роботи реалізації (рисунок виконаний самостійно)

Інтерфейс ClassParserOptions визначає структуру параметрів, які використовуються в процесі парсера. Він містить налаштування, такі як величина відступу, що дозволяє налаштувати форматування коду.

Змінна defaultOptions задає стандартні значення для параметрів парсера, зокрема встановлює відступ на 2 пробіли. Це забезпечує однорідність у форматуванні згенерованого коду. Код, що виконує автоматичну генерацію Solidity-коду, є інструментом, який спрощує процес трансформації класів TypeScript у Solidity. Це дозволяє розробникам легше створювати смарт-контракти, автоматизуючи частину роботи.

Ця ініціатива представляє собою початкову спробу транскомпіляції, демонструючи концепцію перетворення програмного коду з однієї мови в іншу через обробку класів TypeScript.

У майбутньому планується розширення функціональності, що включає підтримку складніших мовних конструкцій, реалізацію перевірки типів, інтеграцію з системами безпеки та поглиблену оптимізацію коду для Ethereum Virtual Machine (EVM). Ці вдосконалення допоможуть зробити інструмент більш потужним і корисним для розробників.

5.1.3 Проведення перевірки та підготовки даних

У контексті дослідження можливостей компіляції TypeScript у Solidity та аналізу відповідного інструментарію, обробка та підготовка даних охоплює низку важливих аспектів, що забезпечують коректну взаємодію між різними мовами та підвищують ефективність розробки. Це включає попередню обробку TypeScript-коду з метою аналізу його структури, побудову абстрактного синтаксичного дерева (AST) для подальшої трансформації, вилучення релевантної інформації для генерації типів або коду на Solidity, а також підготовку згенерованого результату до компіляції офіційним компілятором Solidity. Також важливим етапом є синхронізація даних між результатами обох компіляторів, зокрема узгодження типів, параметрів функцій і структури контрактів для забезпечення типобезпечної інтеграції.

Дослідження можливостей мови програмування TypeScript в контексті розробки смарт-контрактів відкриває нові горизонти для програмістів. TypeScript, завдяки своїй статичній типізації та підтримці сучасних функцій JavaScript, може стати потужним інструментом для створення безпечних і ефективних смарт-контрактів.

Аналіз підходів до реалізації смарт-контрактів на TypeScript виявляє кілька ключових аспектів. По-перше, використання TypeScript дозволяє розробникам з легкістю впроваджувати типи, що сприяє виявленню помилок на етапі компіляції. Це значно підвищує надійність коду і зменшує ризик помилок, які можуть виникнути під час виконання.

По-друге, TypeScript надає можливість використовувати об'єктно-орієнтоване програмування, що дозволяє створювати більш структуровані та зрозумілі рішення. Це особливо важливо для складних смарт-контрактів, де чітка організація коду може полегшити його підтримку та розвиток. Крім того, існують бібліотеки та фреймворки, такі як Hardhat і Truffle, які підтримують TypeScript, що робить процес розробки ще більш зручним. Ці інструменти дозволяють інтегрувати TypeScript у традиційний цикл розробки смарт-контрактів, забезпечуючи розробникам доступ до потужних функцій для тестування, деплою та управління контрактами.

Таким чином, дослідження можливостей TypeScript для створення смарт-контрактів демонструє, що ця мова може стати важливим елементом у розробці безпечних, надійних і масштабованих рішень для блокчейн-технологій.

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів включає в себе аналіз різних підходів до реалізації таких контрактів. Основним етапом є створення абстрактного синтаксичного дерева (AST), яке відображає всі синтаксичні елементи програми. Це дерево дозволяє здійснювати подальшу програмну обробку, аналіз та трансформацію коду. Для цього можна використовувати стандартний компілятор tsc або ж спеціалізовані бібліотеки, такі як ts-morph. Після створення AST виконується семантичний аналіз, що включає перевірку відповідності типів, коректності викликів функцій

та доступності змінних. В результаті цього процесу формується збагачене представлення коду, яке містить усю необхідну інформацію для подальшої трансформації. У випадку, якщо потрібно реалізувати смарт-контракти на TypeScript, важливо виконати додаткову обробку цього представлення - нормалізувати його та перетворити в проміжну форму, яка буде більш адаптованою до цільової мови або віртуальної машини EVM. Цей підхід відкриває нові можливості для інтеграції TypeScript у сферу розробки смарт-контрактів, забезпечуючи зручність та безпеку при програмуванні.

Компіляція коду Solidity виконується за допомогою компілятора solc, що проходить через традиційні етапи: лексичний і синтаксичний аналіз, семантичну перевірку та генерацію машинного коду. В результаті цього процесу формується байт-код, який може виконуватись у віртуальній машині Ethereum, а також створюється ABI - структура у форматі JSON, що описує доступні зовнішні методи, їх параметри та типи.

ABI є ключовим елементом для інтеграції з TypeScript-кодом через інструменти, такі як TypeChain або Abitype, які забезпечують типобезпечну взаємодію з контрактами на рівні TypeScript. Таким чином, обробка вихідного коду на обох мовах не лише забезпечує трансляцію в цільовий код, але й генерує важливі дані для подальшої взаємодії між мовами в рамках Web3-додатків. Це дозволяє розробникам використовувати можливості TypeScript для безпечної та зручної роботи зі смарт-контрактами, забезпечуючи ефективність у розробці та інтеграції.

У сучасній Web3-розробці важливою складовою є підготовка даних для типобезпечної взаємодії між TypeScript і Solidity, що забезпечує надійність та зручність у роботі зі смарт-контрактами. У цьому процесі основним джерелом інформації виступає ABI - JSON-файл, який генерується компілятором Solidity. Цей файл містить опис доступних зовнішніх методів контракту, їх параметрів і типів, що дозволяє розробникам ефективно інтегрувати контракти в TypeScript-код. Завдяки ABI, розробники можуть користуватися перевагами типобезпечного програмування, що зменшує ймовірність помилок і підвищує продуктивність при

взаємодії з блокчейном. Таким чином, правильна підготовка і використання ABI є критично важливими для успішної розробки Web3-додатків.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private data;

    function set(uint256 x) public {
        data = x;
    }

    function get() public view returns (uint256) {
        return data;
    }
}

ABI:
[
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "x",
        "type": "uint256"
      }
    ],
    "name": "set",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "get",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  }
```

Цей код є прикладом смарт-контракту, написаного на мові Solidity, яка використовується для створення децентралізованих додатків на платформі Ethereum. Він починається з директиви, що вказує на ліцензію MIT, під якою розповсюджується код, а також на версію компілятора Solidity 0.8.0 або новішу.

Контракт має назву SimpleStorage і містить змінну стану типу uint256, що є приватною і використовується для зберігання цілого числа.

Контракт включає дві основні функції. Перша функція, set, є публічною і приймає один параметр типу uint256, який називається x. Коли ця функція викликається, значення x присвоюється змінній data, оновлюючи її значення. Друга функція, get, також є публічною, але не має вхідних параметрів. Вона позначена як view, що означає, що вона не змінює стан контракту. Ця функція повертає значення змінної data, що дозволяє користувачам отримувати збережене число.

Для цього контракту також надається ABI (Application Binary Interface), який визначає, як взаємодіяти з контрактом. ABI містить інформацію про функції, їх параметри та типи даних. У даному випадку, ABI описує функцію set, яка приймає параметр x типу uint256 і не повертає значень, а також функцію get, яка не має вхідних параметрів і повертає значення типу uint256. Ця специфікація є важливою для інтеграції контракту з зовнішніми додатками та забезпечує коректне виконання викликів до функцій.

У процесі розробки смарт-контрактів важливою є підготовка та обробка даних для тестування та розгортання. Щоб написати юніт- та інтеграційні тести на TypeScript, розробники повинні заздалегідь підготувати вхідні дані, які будуть передаватися у функції контрактів, визначити очікувані результати виконання, а також створити мок-об'єкти для симуляції зовнішніх залежностей, таких як відповіді від інших контрактів чи API. Ці дані організовуються у TypeScript-файлах, і завдяки системі типізації вони автоматично проходять валідацію до виконання тестів, що значно знижує ймовірність помилок. Для успішного запуску смарт-контрактів важливо належним чином підготувати дані, які потрібні для їх коректної ініціалізації в обраній мережі. Це можуть бути адреси для мережі або специфічних сервісів, приватні ключі, а також доступ до них через захищені змінні середовища. До того ж, важливо визначити параметри, що передаються в конструктор контракту, і налаштувати ліміти газу та його ціну. Усі ці дані зазвичай зберігаються у конфігураційних файлах, таких як .env, hardhat.config.ts

або deploy.config.ts, і завантажуються автоматично під час виконання скриптів для розгортання. Використання TypeScript забезпечує типізацію цих даних, що допомагає підтвердити їх коректність і безпечну обробку в рамках скриптів. Отже, ретельна підготовка даних є вирішальним чинником для стабільного й контрольованого процесу тестування та розгортання смарт-контрактів.

У теоретичному дослідженні можливостей мови програмування TypeScript для розробки смарт-контрактів підготовка даних охоплює кілька ключових етапів, які сприяють створенню ґрунтовної бази знань. На першому етапі проводиться збір літературних джерел, що включає систематичний пошук наукових статей, технічної документації, оглядових матеріалів, публікацій у блогах, а також відкритих репозиторіїв на GitHub, які стосуються тематики TypeScript, Solidity, блокчейн-розробки, абстрактних синтаксичних дерев (AST), проміжних представлень (IR) та формальної верифікації.

Дослідження також передбачає аналіз різних підходів до реалізації смарт-контрактів на мові TypeScript, що включає вивчення існуючих бібліотек, фреймворків та інструментів, які полегшують інтеграцію TypeScript у процес розробки смарт-контрактів. Це дозволяє виявити переваги та недоліки використання TypeScript у контексті блокчейн-розробки, а також визначити потенційні напрямки для подальших досліджень і вдосконалення методів розробки.

У рамках дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів, важливим етапом є обробка зібраних літературних джерел. Цей процес включає категоризацію матеріалів за різними критеріями, такими як тип інструменту, підхід до компіляції, наявність підтримки формальної верифікації та опис обмежень цільової платформи. Така систематизація дозволяє виявити закономірності між різними підходами, а також визначити прогалини у наявних рішеннях, що потребують подальшого вивчення.

На етапі синтезу здійснюється інтеграція отриманої інформації з різних джерел, що сприяє формуванню узагальненого уявлення про теоретичний ландшафт предметної області. Візуалізація результатів цього процесу у вигляді

таблиць, діаграм та графіків допомагає проілюструвати складні взаємозв'язки, підкреслити ключові переваги та недоліки різних підходів до реалізації смарт-контрактів на мові TypeScript, а також систематизувати великий обсяг матеріалів для подальшого аналізу. Це, в свою чергу, забезпечує глибше розуміння потенціалу TypeScript у контексті розробки смарт-контрактів та відкриває нові перспективи для подальших досліджень у цій галузі.

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів є важливим аспектом сучасного програмування, що охоплює різноманітні підходи до реалізації смарт-контрактів. Аналіз цих підходів включає повний цикл обробки даних, починаючи з парсингу вихідного коду, побудови абстрактних синтаксичних дерев (AST), виконання семантичного аналізу, генерації проміжного або цільового представлення (наприклад, ABI), і завершуючи формуванням типізованих інтерфейсів для безпечної інтеграції з фронтендом або тестовими середовищами.

Ці процеси є основою для розробки сучасних інструментів, які реалізують "паралельну компіляцію", що дозволяє інтегрувати різні мовні екосистеми в єдиний робочий процес. Завдяки такій інтеграції, розробники отримують можливість більш ефективно та безпечно працювати з кодом смарт-контрактів, знижуючи ризики помилок і підвищуючи якість кінцевого продукту. Аналіз різних підходів до реалізації смарт-контрактів на TypeScript дозволяє виявити переваги та недоліки існуючих рішень, а також окреслити нові напрямки для подальших досліджень у цій галузі.

5.2 Методи порівняння за критерієм швидкості компіляції

Оцінка переваг використання мови TypeScript для розробки смарт-контрактів неможлива без врахування важливого критерію - швидкості компіляції. Цей фактор суттєво впливає на зручність процесу розробки, тривалість циклів створення програмного забезпечення та загальну продуктивність команди. У рамках дослідження була обрана методика, яка дозволяє виміряти та порівняти час компіляції для різних методів створення

контрактів - як традиційного підходу з використанням Solidity, так і застосування TypeScript як первинної мови програмування з подальшим трансформуванням коду в Solidity.

Для об'єктивного оцінювання часу компіляції були розроблені смарт-контракти з однаковою логікою, реалізовані двома різними способами: традиційно за допомогою Solidity та через TypeScript, який трансформується в Solidity за допомогою інструментів, таких як TypeChain, плагіни Hardhat та інші сучасні рішення. Під час експерименту було зафіксовано загальний час, необхідний для перетворення TypeScript-коду в проміжні артефакти (такі як ABI або JSON-інтерфейси), трансляції в Solidity (де це було необхідно), а також подальшої компіляції в байт-код за допомогою компілятора Solidity. Для обох методів вимірювання часу компіляції проводилось в контрольованих однакових умовах, що дозволило уникнути спотворень у результатах, викликаних зовнішніми факторами, такими як продуктивність обладнання або активні фонові процеси.

Особливу увагу було звернено на вплив структури проєкту на швидкість компіляції. Зокрема, було досліджено, як тривалість компіляційного процесу змінюється в залежності від масштабування коду, додавання нових смарт-контрактів або оновлення бібліотек. Важливо відзначити, що при використанні TypeScript компіляція включає кілька додаткових етапів, таких як генерація типів, створення обгорток для функцій контракту та налаштування середовища. Це може призвести до незначного збільшення загального часу компіляції, але водночас дозволяє зменшити ризик помилок на наступних етапах, оскільки частина перевірок виконується ще до моменту трансляції в Solidity.

У рамках дослідження також були розглянуті можливості паралельної обробки результатів попередньої компіляції, застосування інкрементальної трансляції та інші методи оптимізації, які активно використовуються в екосистемі TypeScript. Було виявлено, що ці механізми можуть частково компенсувати початкові витрати часу, що сприяє більш швидкому збиранню повторно компільованих проєктів під час тривалих циклів розробки. Це дозволяє

розробникам зосередитися на покращенні функціоналу, зменшуючи час очікування на компіляцію.

В цілому, метод аналізу швидкості компіляції дав змогу дійти висновку, що використання TypeScript для розробки смарт-контрактів є обґрунтованим, якщо забезпечена активна підтримка сучасних інструментів. Додаткові витрати часу на компіляцію виправдовуються покращеною перевіркою типів, зручнішою інтеграцією з фронтенд-технологіями та високим рівнем автоматизації процесів. Хоча розробка на Solidity залишається швидшою при мінімальних навантаженнях, у випадках масштабних проєктів та складної архітектури переваги TypeScript можуть стати вирішальними.

5.3 Методи порівняння за критерієм безпечності у використані

Безпека є критично важливим елементом у процесі розробки смарт-контрактів, оскільки навіть найменші помилки можуть призвести до серйозних фінансових втрат або підриву довіри до всієї екосистеми блокчейн. У зв'язку з цим, у рамках дослідження потенціалу TypeScript для написання смарт-контрактів особлива увага приділяється аналізу безпеки. Це включає в себе порівняння різних підходів з точки зору їхньої надійності та стійкості до атак, щоб забезпечити максимальний захист для користувачів і інвесторів.

Аналіз ґрунтується на всебічній методології, що поєднує теоретичні основи виявлення типових ризиків з практичною перевіркою на реальних прикладах. Спочатку було акцентовано увагу на типобезпеці, яку забезпечує TypeScript. Завдяки своїй статичній системі типів, мова дозволяє виявляти численні помилки ще до моменту виконання коду. В процесі дослідження було оцінено відповідність типів між описами смарт-контрактів у TypeScript і їх реалізаціями в Solidity, а також проаналізовано вплив автоматичної генерації інтерфейсів (ABI) на точність типізації та безпеку обміну даними.

Для оцінки згенерованих смарт-контрактів були використані статичні аналізатори безпеки, такі як Slither і Mythril. Ці інструменти дозволили виявити наявність поширених вразливостей, таких як повторні виклики функцій

(reentrancy), переповнення змінних (integer overflow/underflow), залежності від часу, а також інші потенційно небезпечні фрагменти коду. Додатково проводився аналіз обсягу та якості коду, що отримується в результаті компіляції з TypeScript, з акцентом на виявлення дублювання логіки, зайвих інструкцій і структур, які можуть негативно вплинути на загальну надійність контракту.

Додатково було проведено тестування функціональності контрактів в рамках контрольованих сценаріїв. Для цього були розроблені юніт-тести та інтеграційні тести, реалізовані за допомогою фреймворків Hardhat і Chai. Це дозволило змоделювати ситуації з некоректними даними, атакуючими викликами та крайніми значеннями. Результати тестування продемонстрували, наскільки стабільно система реагує на аномальні ситуації і чи ефективно працюють механізми валідації вхідних даних.

Оцінка інструментів, що використовуються в зв'язку між TypeScript та Solidity, також проводилася з акцентом на безпеку. Зокрема, було досліджено, наскільки ефективно такі інструменти, як TypeChain та спеціалізовані плагіни для Hardhat, здатні підтримувати типобезпеку під час трансляції, контролювати доступ до функцій смарт-контрактів і забезпечувати інтеграцію з інструментами для аудиту або перевірки вразливостей. Завершальним етапом дослідження стала експертна оцінка ризиків, що проводилася за участі досвідчених розробників смарт-контрактів. Їхні коментарі та зауваження допомогли зіставити результати аналізу з реальними умовами розробки в середовищі Web3 та DeFi, а також надали цінну інформацію щодо потенційних загроз, які можуть залишатися непоміченими під час традиційного тестування.

В цілому, методика оцінки за критерієм безпеки дозволила всебічно проаналізувати доцільність застосування TypeScript у розробці смарт-контрактів, зокрема в контексті їх компіляції в Solidity. Використані підходи включали перевірку якості коду, стабільності виконання, відповідності стандартам безпеки та захисту від найбільш розповсюджених вразливостей. Отримані результати формують надійну основу для висновків про ефективність інтеграції TypeScript у процес створення безпечних і надійних смарт-контрактів.

5.4 Методи роботи методу паралельної компіляція файлів typescript у Solidity

У цьому підрозділі розглядається перспектива покращення швидкості компіляції TypeScript-коду в Solidity-контракти шляхом впровадження паралельної обробки вихідних файлів із використанням API платформи Node.js. Цей підхід є особливо важливим у ситуаціях, коли проект містить велику кількість модулів або контрактів, які зберігаються в окремих файлах і потребують автономної компіляції.Цей підхід є особливо корисним у ситуаціях, коли проект містить велику кількість модулів або контрактів, які зберігаються як окремі файли і потребують незалежної компіляції. Основна ідея полягає в одночасному виконанні кількох процесів компіляції, де кожен процес обробляє свій власний TypeScript-файл, перетворюючи його на відповідний Solidity-контракт. Node.js як серверна платформа пропонує зручні та ефективні механізми для управління потоками і процесами, використовуючи вбудовані модулі, такі як child_process, worker_threads, а також API на зразок Promise.all, що дозволяють виконувати паралельні завдання без блокування основного потоку виконання.

Застосування цього методу дозволяє значно зменшити загальний час компіляції в порівнянні з традиційним послідовним підходом, особливо в умовах використання багатоядерних процесорів. У рамках дослідження був створений прототип компілятора, який обробляє масив TypeScript-файлів, ініціюючи окремий процес трансляції для кожного файлу. Після завершення компіляції результати об'єднуються в єдину структуру, що забезпечує подальше використання в екосистемі Ethereum. Ця реалізація не лише продемонструвала стабільність і масштабованість, але й включала механізми моніторингу, які дозволяють відстежувати продуктивність кожного процесу в реальному часі, що ще більше підвищує ефективність роботи з великими обсягами даних. Таким чином, система адаптується до змін у навантаженні, оптимізуючи використання ресурсів. Додатково, система забезпечила можливість детального логування помилок для кожного окремого файлу, що дозволяє розробникам швидко виявляти та усувати проблеми без переривання загального процесу компіляції. Це

є критично важливим у контексті командної розробки великих децентралізованих застосунків, де одночасно працює кілька учасників. Більше того, інтеграція з системами безперервної інтеграції (CI) дозволила автоматизувати процес тестування та валідації коду, що підвищує якість кінцевого продукту. Завдяки цій функціональності, команда може оперативно реагувати на зміни, зменшуючи ризик виникнення критичних помилок у фінальному релізі.

У ході дослідження було також проведено комплексний аналіз продуктивності різних моделей паралельного виконання. Встановлено, що використання worker-потоків дозволяє значно знизити споживання пам'яті в порівнянні з традиційним запуском дочірніх процесів. Однак, класичний підхід продемонстрував вищу стабільність при одночасній компіляції великої кількості контрактів. Окрім того, були протестовані стратегії оптимізації через кешування проміжних результатів, що дало змогу уникнути повторної трансляції незмінних файлів, ще більше зменшуючи навантаження на систему. Цікаво, що також було виявлено, що впровадження механізмів адаптивного управління ресурсами в реальному часі покращує загальну продуктивність, дозволяючи системі динамічно реагувати на зміни в обсязі вхідних даних і, таким чином, підвищувати ефективність роботи в умовах високих навантажень.

Узагалі, впровадження паралельної компіляції за допомогою інструментів Node.js виявилося ефективним у реальних умовах розробки, сприяючи не лише значному збільшенню швидкості обробки, а й покращенню адаптивності у формуванні власних пайплайнів для обробки коду. Цей підхід дозволяє налаштовувати процес трансляції відповідно до специфіки кожного проекту, що робить його легким для масштабування у разі збільшення обсягу контрактів. Крім того, інтеграція таких рішень у CI/CD-системи відкриває нові можливості для автоматизації тестування та деплою, що підвищує надійність розробки. Важливо зазначити, що використання таких сучасних технологій, як Docker для контейнеризації середовища виконання, дозволяє ще більше спростити процес налаштування та забезпечити консистентність між різними етапами розробки, що

робить цей підхід ще більш привабливим для розробників смарт-контрактів на базі TypeScript.

Для кращого розуміння отриманих результатів було оформлено рисунок – 5, в якій представлено порівняння середнього часу компіляції в залежності від кількості оброблюваних TypeScript-файлів. У таблиці видно, що паралельна компіляція демонструє значні переваги у швидкості порівняно з послідовною компіляцією.

Також слід зазначити , що повну версію алгоритму паралельного компілювання TypeScript коду в Solidity код можна побачити в додатку Б.

Кількість TypeScript файлів	Час компіляції (секунди)	Паралельна компіляція (секунди)	Зменшення часу (%)
10	60	20	66.67
20	120	40	66.67
50	300	75	75.00
100	600	150	75.00
200	1200	250	79.17
500	3000	600	80.00

Рисунок 5 – Порівняння послідовного та паралельного алгоритму під час компіляції Typescript в Solidity (рисунок виконаний самостійно)

Ці дані підтверджують, що при збільшенні обсягу файлів вигоди від паралельної компіляції стають ще більш помітними, що робить цей підхід надзвичайно ефективним для великих проектів. Додатково, важливо зазначити, що паралельна компіляція не лише скорочує час обробки, але й дозволяє розробникам зосередитися на інших важливих аспектах проекту, таких як тестування та оптимізація коду. Це, в свою чергу, підвищує загальну продуктивність команди та якість кінцевого продукту.

Як демонструє таблиця, з ростом кількості вхідних файлів ефективність паралельної компіляції значно зростає, що дозволяє досягти майже трьохразового скорочення часу в порівнянні з традиційним послідовним методом. Ця різниця стає особливо помітною при роботі з великими проектами, які включають десятки або навіть сотні смарт-контрактів.

Крім того, паралельна компіляція не лише підвищує швидкість обробки, але й зменшує навантаження на розробників, дозволяючи їм зосередитися на інших критичних задачах, таких як тестування, аудит безпеки та оптимізація коду. Це, в свою чергу, сприяє підвищенню загальної якості продукту та зменшенню ризиків, пов'язаних із помилками в коді.

ВИСНОВКИ

Під час проведення дослідження зі сфери можливостей мови програмування TypeScript для створення смарт-контрактів та аналізу шляхів покращення інструментів розробки в контексті компіляції TypeScript в Solidity, було виявлено кілька ключових аспектів, які впливають на ефективність і перспективність використання TypeScript у блокчейн-екосистемі. Аналіз цих аспектів показав, що, незважаючи на наявні труднощі, поєднання TypeScript із Solidity має значний потенціал для спрощення процесу розробки смарт-контрактів і підвищення їхньої безпеки.

Під час дослідження особливий спектр уваги слід приділяти можливостям TypeScript у розробці смарт-контрактів. Завдяки своїй здатності забезпечувати строгість типів, TypeScript є відмінним вибором для створення смарт-контрактів, особливо в контексті розробки децентралізованих додатків (DApp) на основі блокчейнів, таких як Ethereum. Система типізації допомагає зменшити ризик помилок, пов'язаних із некоректним використанням типів даних, що є надзвичайно важливим у середовищі блокчейн, де навіть незначна помилка може призвести до фінансових втрат або збоїв у системі. Використання TypeScript дозволяє розробникам ефективно створювати типізовані інтерфейси для взаємодії з контрактами, що значно знижує ймовірність помилок під час виконання транзакцій і роботи з блокчейном.

Поєднання TypeScript із Solidity. Хоча Solidity є провідною мовою для розробки смарт-контрактів на платформі Ethereum, TypeScript не має широкої підтримки для безпосереднього написання таких контрактів. Відсутність прямої інтеграції між TypeScript і Solidity ускладнює цей процес. Проте, завдяки наявності різноманітних інструментів і бібліотек, таких як TypeChain, Web3.js та Hardhat, взаємодія між TypeScript і Solidity може бути значно спрощена. TypeChain надає можливість створювати типізовані клієнтські бібліотеки для взаємодії з контрактами, що знижує ризик виникнення помилок під час виконання транзакцій або виклику функцій контрактів. Це дозволяє програмістам зосередитися на бізнес-логіці, зменшуючи при цьому занепокоєння щодо проблем

із типами та помилок компіляції. Hardhat забезпечує можливість тестування смарт-контрактів прямо в середовищі TypeScript, що дозволяє розробникам створювати зручні робочі процеси, адаптовані для тих, хто звик працювати з цією мовою програмування.

Теоретичний аналіз вказує на серйозні труднощі, пов'язані з розробкою повноцінного компілятора, який би перетворював TypeScript безпосередньо в Solidity або інший формат, сумісний з Ethereum Virtual Machine (EVM). Головною проблемою є глибокий семантичний розрив між цими мовами програмування. TypeScript, як мова, орієнтована на динамічні вебдодатки, має гнучку систему типів, що дозволяє розробникам працювати з різноманітними структурами даних і динамічно змінювати їх у процесі виконання програми. Вона функціонує в середовищі з автоматичним управлінням пам'яттю, що забезпечує зручність і безпеку при розробці складних застосунків.

На відміну від цього, Solidity є строго типізованою мовою з обмеженим набором конструкцій, що спеціально розроблена для написання смарт-контрактів на блокчейні Ethereum. Також слід зазначити , що База даних в TypeScript зазвичай використовується для роботи з даними в додатках, написаних на TypeScript, і може бути реалізована через різні бібліотеки та фреймворками , як приклад можна використовувати інформацію с джерела [11].Тож вона надає прямий доступ до ресурсів EVM, що є критично важливим для оптимізації виконання коду в умовах обмежених ресурсів. Цей семантичний розрив призводить до того, що багато концепцій, притаманних TypeScript, не можуть бути безпосередньо перенесені в Solidity без значних адаптацій. Таким чином, розробка компілятора, який б ефективно трансформував код з TypeScript у Solidity, стикається з численними викликами, що потребують глибокого розуміння обох мов та їхніх особливостей.

Поєднання TypeScript із Solidity. Хоча Solidity є провідною мовою для розробки смарт-контрактів на платформі). Проте слід підкреслити також і труднощі при поєднанні Solidity та TypeScript , наприклад : Solidity є мовою нижчого рівня, і її типи даних суттєво відрізняються від тих, що

використовуються в TypeScript, що ускладнює коректну компіляцію та передбачуваність результатів, хоча Solidity дозволяє створювати код, що функціонує на рівні блокчейну, він може бути підданий ризику атак, таких як переповнення або недостатня перевірка даних. Використання TypeScript може допомогти зменшити кількість подібних помилок на етапі розробки, проте проблема залишається актуальною, оскільки Solidity має свої специфічні особливості, які важко узгодити з загальноприйнятими стандартами типізації.

І дивлячись на це саме тому існує необхідність у створенні нових компіляторів або інструментів, які здатні забезпечити більш зрозумілу та безпечну взаємодію між TypeScript і Solidity, враховуючи всі нюанси сумісності типів.

Перспективи розвитку інструментів і стандартів. Ясно, що існує потреба у вдосконаленні нових стандартів і бібліотек для покращення інтеграції між TypeScript і Solidity. Протоколи для генерації типів для смарт-контрактів, а також бібліотеки для автоматизованого тестування і верифікації контрактів можуть значно підвищити ефективність розробки та зменшити кількість помилок під час тестування.

Окрім цього, з ростом популярності таких технологій, як DeFi, NFT та інших ініціатив у сфері блокчейн, виникає потреба у розробці інструментів, які спростять інтеграцію цих технологій у щоденну практику розробки. Це дозволить програмістам швидко адаптувати свої інтерфейси, підвищити рівень безпеки смарт-контрактів і знизити ймовірність виникнення критичних помилок.

Застосування типобезпечності для посилення рівня безпеки. Однією з ключових переваг TypeScript є його здатність забезпечувати типобезпечність, що суттєво зменшує ймовірність виникнення помилок, пов'язаних із некоректним використанням типів, що є особливо актуальним у сфері розробки смарт-контрактів. З огляду на специфіку блокчейн-систем та критичну важливість точного контролю за виконанням контрактів, впровадження типобезпечності на стадії розробки дозволяє виявляти серйозні помилки на ранніх етапах, ще до їх запуску в основній мережі. Це, в свою чергу, сприяє зменшенню ризиків

виникнення вразливостей і атак, таких як переповнення буфера або помилки, що виникають через неправильну обробку даних.

Невизначені можливості для розвитку інтеграції. Поєднання TypeScript і Solidity є важливим елементом еволюції екосистеми блокчейн-розробки, проте цей процес ускладнюється через брак стандартів і відсутність нативної підтримки в Solidity. У майбутньому створення нових компіляторів, вдосконалених бібліотек та стандартів може забезпечити більш ефективну та безпечну взаємодію між цими мовами програмування. Існує ймовірність, що з'являться нові інструменти, які автоматизують процес переходу від TypeScript до Solidity, що зробить його більш зрозумілим та зручним для розробників.

TypeScript демонструє значний потенціал у сфері розробки смарт-контрактів, оскільки здатний підвищити рівень типобезпечності та зменшити ймовірність виникнення помилок під час програмування. Співпраця між TypeScript і Solidity, за допомогою таких інструментів, як TypeChain і Hardhat, істотно спрощує процес тестування та створення смарт-контрактів. Виклики, що виникають через відмінності в типах даних і брак нативної підтримки, вимагають створення нових інструментів і стандартів для покращення інтеграції.

Подальші дослідження та створення нових інструментів можуть суттєво поліпшити процес інтеграції між цими мовами програмування, сприяючи їх більш ефективному застосуванню в сфері блокчейн-розробки. Отже, застосування TypeScript у контексті блокчейн-розробки, особливо при створенні смарт-контрактів, може значно підвищити ефективність розробки та тестування. Проте для досягнення повної інтеграції з Solidity необхідно вирішити ряд технічних і концептуальних викликів.

Дослідження можливостей мови програмування TypeScript для розробки смарт-контрактів виявило значний потенціал цієї мови в контексті сучасних технологій блокчейн. Типізація, яку пропонує TypeScript, дозволяє зменшити ризики помилок на етапі компіляції, що є критично важливим фактором для розробки смарт-контрактів, де безпека є пріоритетом. Крім того, TypeScript легко інтегрується з популярними бібліотеками та фреймворками, такими як Hardhat та

Truffle, що спрощує процес розробки та тестування смарт-контрактів. Використання паралельної компіляції також дозволяє об'єднувати різні мови програмування в єдину екосистему, що підвищує ефективність роботи та зменшує час на розробку.

Практичний аспект розробки смарт-контрактів на TypeScript також демонструє значні переваги. Завдяки можливостям цієї мови, розробники можуть швидше створювати та тестувати смарт-контракти, що позитивно впливає на загальну продуктивність проектів. Використання типізованих інтерфейсів та семантичного аналізу допомагає виявляти помилки на ранніх етапах, що знижує ризики при впровадженні смарт-контрактів у реальні проекти. Крім того, TypeScript надає можливість легко масштабувати проекти, що є важливим для розробки складних систем, які потребують інтеграції з різними блокчейн-протоколами.

Загалом, дослідження можливостей TypeScript для розробки смарт-контрактів показало, що ця мова є потужним інструментом, здатним забезпечити безпеку, ефективність і гнучкість у процесі розробки. Подальші дослідження можуть зосередитися на вдосконаленні існуючих підходів та розробці нових рішень, які ще більше полегшать інтеграцію TypeScript у світ блокчейн-технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. QuickNode AssemblyScript for Blockchain Development, "/ arxiv 2023, URL: https://marketplace.quicknode.com/ (дата звернення: 09.18.2024).
2. Бібліотека JavaScript/TypeScript для взаємодії з Ethernum "/ arxiv 2023, URL: https://web3js.readthedocs.io/ (дата звернення: 10.12.2024).
3. USING GUIDE WITH NEAR Protocol Documentation "/arxiv, 2022, URL: https://www.near-sdk.io/ (дата звернення: 12.12.2024)
4. Ethers.js Documentation "/arxiv, 2023, URL: https://docs.ethers.io/v5/ (дата звернення: 18.12.2024)
5. Typechain"/arxiv,2025,URL: https://www.alchemy.com/dapps/typechain (дата звернення: 05.01.2025)
6. Solidity USING EXPERIENCE documentation "/ arxiv, 2023, URL: https://docs.soliditylang.org/ (дата звернення: 06.01.2025)
7. Typechain у сучасності. (підручник з теорії компіляції та реалізації "/ SHRIMAN KR, 2022,37с .
8. Приклади методів розробки з Typechain від elatru URL: https://elartu.tntu.edu.ua/handle/lib/36536 (дата звернення 17.12.2024)
9. Вивчення використання блокчейну та смарт-контрактів за допомогою мови програмування Rust для покращення рішень у різних сферах URL: https://elartu.tntu.edu.ua/handle/lib/43276 (дата звернення 12.16.2024)
10. Afanasieva, I., Golian, N., Golian, V., Khovrat, A., & Onyshchenko, K. (2023). Application of Neural Networks to Identify of Fake News. COLINS (2), 346-358.
11. Mazurova, O., Naboka, A., & Shirokopetleva, M. (2021). Research of ACID Transaction Implementation Methods for Distributed Databases Using Replication Technology. Innovative Technologies and Scientific Solutions for Industries, 2(16), 45–57, URL: https://doi.org/10.30837/itssi.2021.16.019. (дата звернення 12.18.2024)

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

10. Afanasieva, I., Golian, N., Golian, V., Khovrat, A., & Onyshchenko, K. (2023). Application of Neural Networks to Identify of Fake News. COLINS (2), 346-358.

11. Mazurova, O., Naboka, A., & Shirokopetleva, M. (2021). Research of ACID Transaction Implementation Methods for Distributed Databases Using Replication Technology. Innovative Technologies and Scientific Solutions for Industries, 2(16), 45–57, URL: https://doi.org/10.30837/itssi.2021.16.019. (дата звернення 12.18.2024).