МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

РОГОЖУК НАЗАР ВЯЧЕСЛАВОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

РОЗРОБКА І ДОСЛІДЖЕННЯ ДЕЦЕНТРАЛІЗОВАНОГО МЕДІА ДОДАТКУ НА БАЗІ ТЕХНОЛОГІЇ BLOCKCHAIN

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Р.М. Бабаков, доцент кафедри
інформаційних технологій
д.т.н., доцент,
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Рогожук Н. В. Розробка децентралізованого медіа додатку на базі технології Blockchain. Спеціальність 122 «Ком’ютерні науки», освітня програма «Комп’ютерні технології обробки даних» Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній роботі досліджені методи побудови децентралізованих дедаток з використанням сучасних інструментів та підходів. Розглянуто існуючі програмні сервіси з подібним функцоналом, та реалізовано власне рішення децентралізованого додатку, котре надає платформу для розміщення задач спортивного програмування з можливістю вирішувати надсилати рішення та отримувати винагороду за правильні розв’язки.

Ключові слова: децантралізований додаток, блокчейн, ефіріум, архітектура, сучасні технології, розробка.

65 с., 31 рис., 15 джерел.

ABSTRACT

Rogozhuk N. V. Development of a decentralized media application based on Blockchain technology. Specialty 122 "Computer Science", educational program "Computer Data Processing Technologies" Vasyl' Stus Donetsk National University, Vinnytsia, 2022.

The qualification work investigates methods of building decentralized dedatas using modern tools and approaches. Existing software services with similar functionality are considered, and our own solution of a decentralized application is implemented, which provides a platform for placing sports programming tasks with the ability to send solutions and receive rewards for correct solutions.

Keywords: decentralized application, blockchain, ethereum, architecture, modern technologies, development.

Pages 65, figure 31, sources 15 sources.

ЗМСТ

ВСТУП

З кожним днем розробка додатків з використанням технології блокчейн стає все популярнішою, людям це цікаво, і найголовніше потрібно. Потрібно для того щоб відчувати себе в безпеці, анонімності та впевненості. Комусь це цікаво з метою заробити коштів, використовуючи біржі, комусь як інструмент для платежів, комусь для забавки. І насправді сфер використання є необмежена кількість. І з кожним днем цих сфер тільки збільшується.

Тема магістерської роботи – «Розробка децентралізованого медіа додатку на базі технології Blockchain». Якщо більш конкретизувати тему, то мова йде про розробку навчального додатку на базі технології блокчейн з використанням смарт контрактів, підходу невзаємозамінного токену. Основною ідеєю, є що кінцевий користувач може реалізувавши певний протокол токену, завантажити умову задачі спортивного програмування та тестові дані до мережі ефіріум, і також інші користувачі, можуть спробувати розв’язати задачу, і за кожен пройдений елемент з тестових даних з першого разу отримати певну винагороду.

Також неможливо обійтись без використання стандартного підходу побудови клієнт серверної архітектури при реалізації даного додатку. Отже визначимось з метою роботи:

- Огляд існуючих аналогів, котрі мають подібний набір технологій
- Ознайомлення з алгоритмами, котрі лежать у фундаменті технології блокчейн
- Реалізація частини децентралізованого додатку з можливістю зберігання
- Побудова архітектури, яка буде легко масштабуватись горизонтально
- Реалізація смарт контракту в мережі ефіріум

- Реалізація клієнт серверної частини з використанням підходу web 3.0

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1 Актуальність теми

Сфера блокчейну та криптовалют є відносно молодою, і Україна в розробці різноманітних проектів та інтеграції їх в повсякденне життя займає далеко не останнє місце. З кожним роком запускають нові українські стартапи, які стають популярними у всьому світі. Також для світових гігантів Україна є досить цікавою країною в контексті інтеграцій з бізнесами.

У 2020 році американська компанія Chainanalysis оприлюднила рейтинг прийняття криптовалют у світі, в рейтингу було задіяно 154 країни, місце у рейтингу залежало від відсоткового значення активних користувачів та обсяг ринків. І Україна в цьому рейтингу зайняла перше місце. І з роками Україна і далі тримається у перших позиціях.

По суті блокчейн – особливий цифровий контракт, за допомогою якого конкретна особа безпосередньо здійснює транзакцію з іншою особою і виставляє їй рахунок. У цьому випадку інформація про транзакції зберігається в комп'ютерній мережі, яка включає в себе комп'ютер покупця і постачальника, що здійснюють транзакцію, в тому числі комп'ютери інших учасників мережі. Банк, як традиційний посередник угод, для даної моделі не потрібен, свідками кожної транзакції між постачальником і покупцем виступають інші учасники мережі, які в змозі підтвердити деталі транзакції,

Говорячи про мережу блокчейнів, неможливо ігнорувати тему безпеки. Захист реєстру транзакцій у технології блокчейну використовує вдосконалені криптографічні алгоритми для захисту від редагування даних та несанкціонованого доступу. Теоретично можна зламати криптографію, але на практиці потрібен комп'ютер з величезними обчислювальними

можливостями. Як правило, блокчейн асоціюється або навіть ототожнюється з криптовалютами, але варто знати, що сфера використання цієї технології набагато ширша і криптовалюти - це лише вершина айсберга. На додаток до широко зрозумілої технологічної та фінансової галузі, цю технологію можна знайти в державному управлінні або в енергетичному секторі.

Отже розробка сервісу, котрий буде взаємодіяти з блокчейном, та давати можливість кінцевим користувачам отримувати певну винагороду за розв'язання задач спортивного програмування є іноваційним, цікавим та корисним.

1.2 Огляд сервісів які мають подібну концепцію

1.2.1 Визначення критеріїв по яким будуть шукатись аналоги

Безпосередньо сервісів, які одночасно базуються на блокчейні та дають змогу користувачам отримувати винагороду за розв’язання задач немає, проте є сервіси які суміжні певним чином.

Концепція сервісу подібна до існуючої концепції Play2Earn. Суть якої полягає в тому, що є можливість граючи в гру, отримувати якісь віртуальні речі, які мають також певну цінність, і використовуючи їх можна більш легко перемагати противників, чи швидше пересуватись по сюжетній лінії.

1.2.2 LeetCode

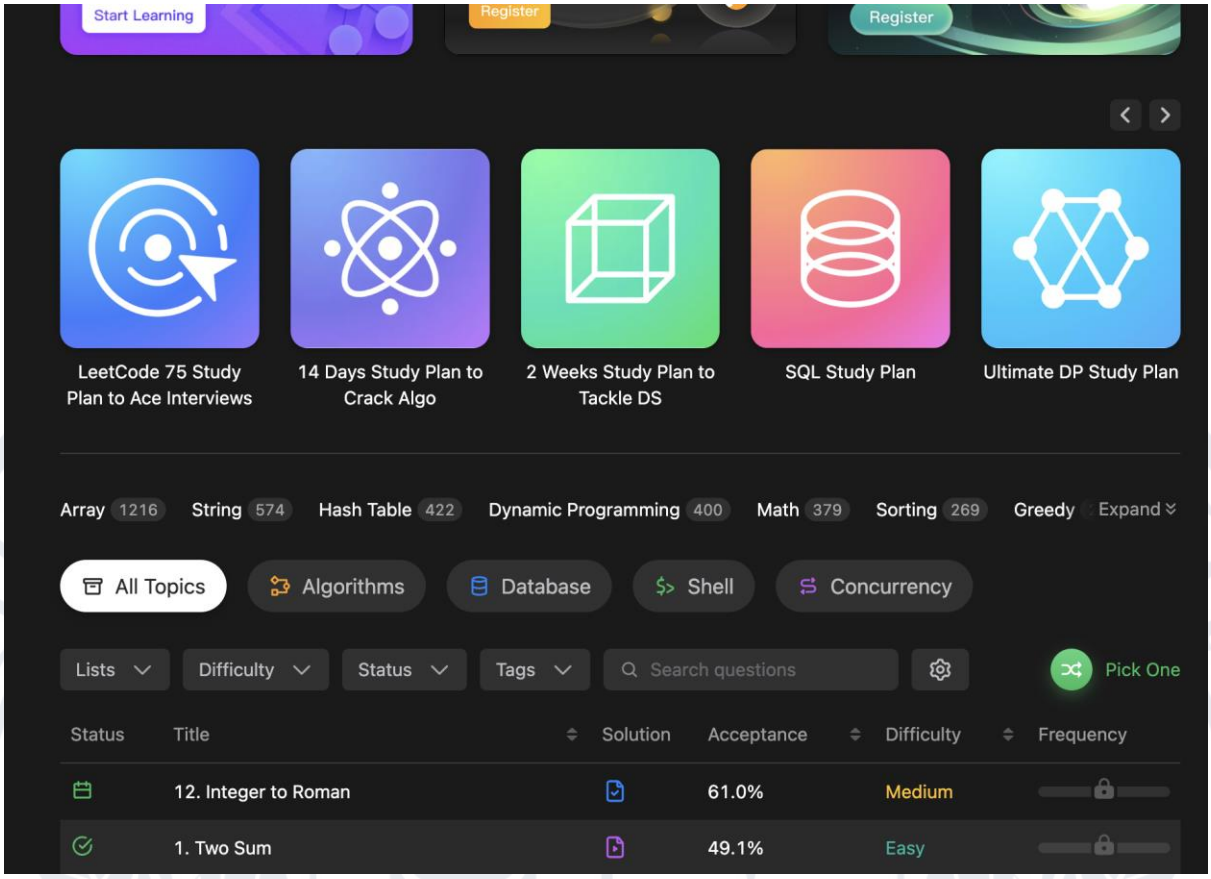


Рисунок 1.1 – огляд LeetCode

Leetcode – є одним з найпопулярніших сервісів де можна розв’язувати олімпіадні задачі, і не тільки олімпіадні. Також регулярно проводять турніри, де за призові місця відправляють брендовані футболки чи кофти. Має велику базу задач, та в принципі, велику кількість користувачів. Також часто використовується компаніями при проведенні співбесід.

1.2.3 HackerRank

HackRank дуже подібний сервіс до LeetCode, але зроблений сильний акцент, саме на пошук роботи. На сервісі можна зареєструватись як і кандидат, так і як, той хто шукає робітника у штат. І відповідно шукаючи кандидата відразу можна зрозуміти якого він рівня, та на що здатний, в контексті алгоритмічного мислення та написання оптимізованого коду.

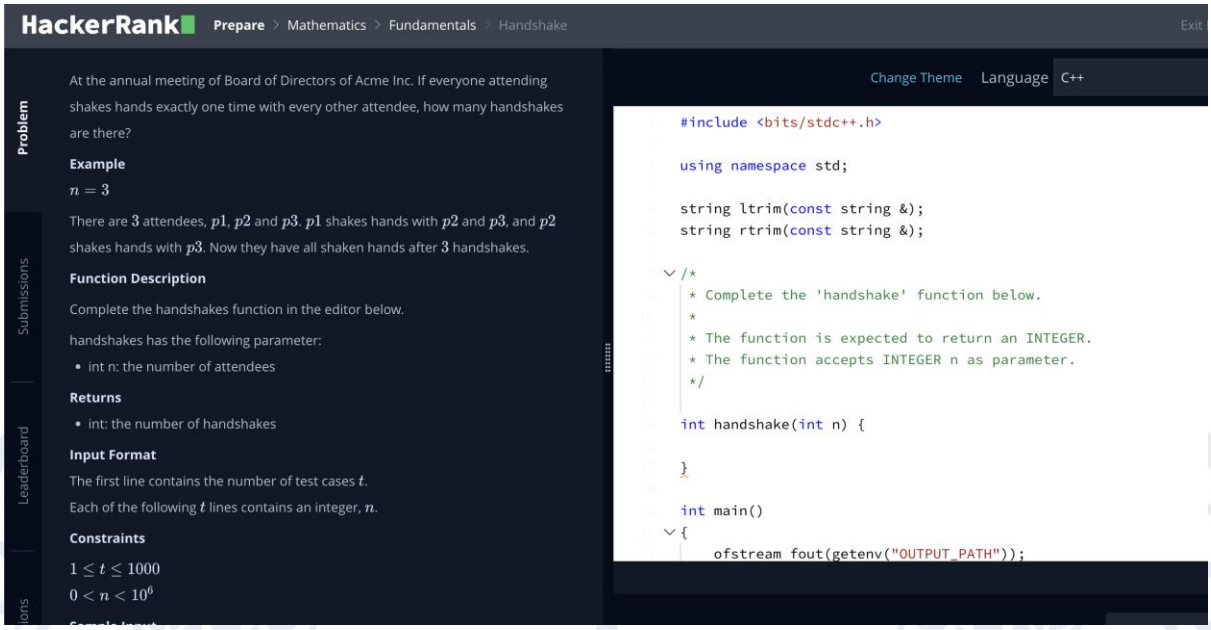


Рисунок 1.2 – огляд HackerRank

1.2.4 OpenSea

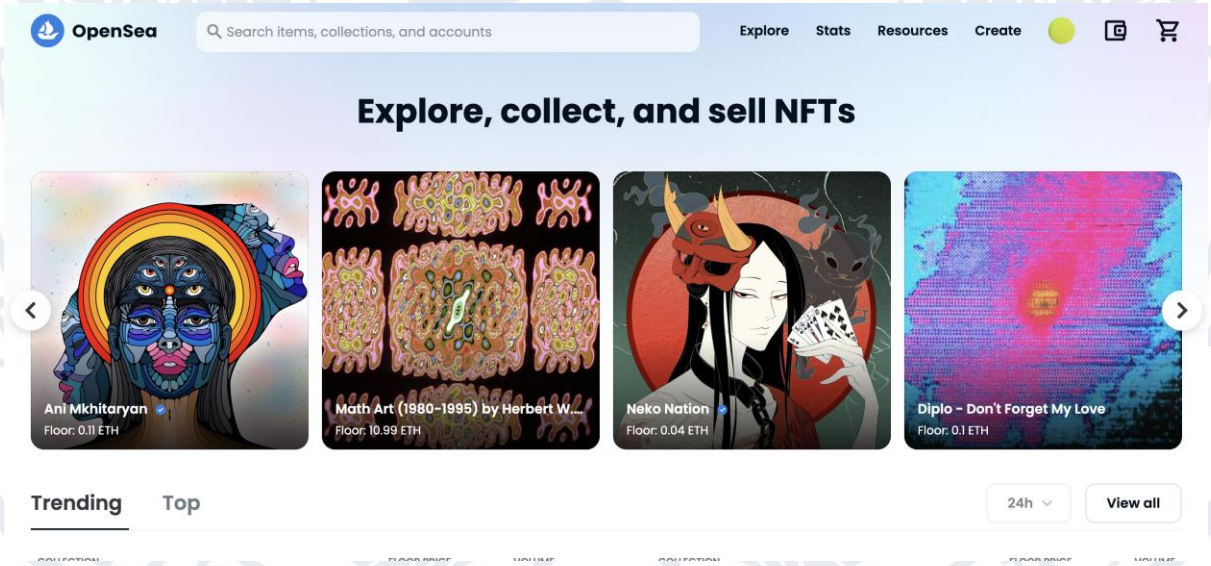


Рисунок 1.3 – огляд OpenSea

OpenSea – це маркетплейс, власне, невзаємозамінний токенів (NFT). Ці NFT повинні реалізовувати певний протокол, за рахунок чого, відображається колекція, або картинок, або інших медіа матеріалів. Користувачі можуть обмінюватись токенами, купувати, торгуватись. Та

просто колекціонувати та збирати колекцію. Відповідно сервіс використовую саме те, що і потрібно для виконання магістерської роботи, а саме, підхід web 3.0, блокчейн, смарт контракти.

1.2.5 Uniswap / Pancakeswap

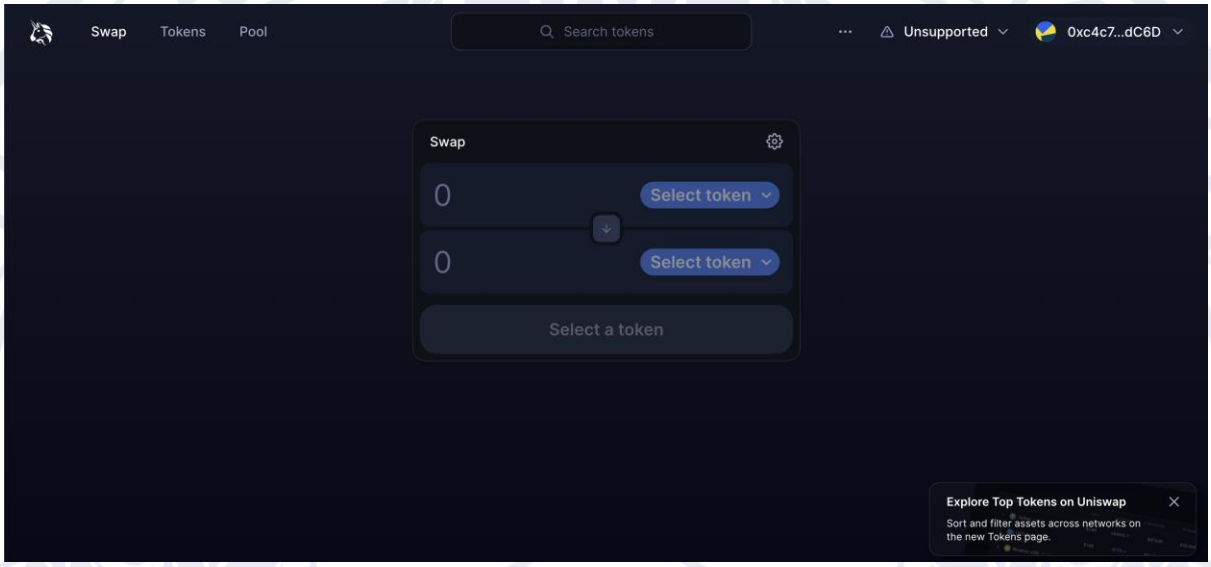


Рисунок 1.4 – огляд Uniswap

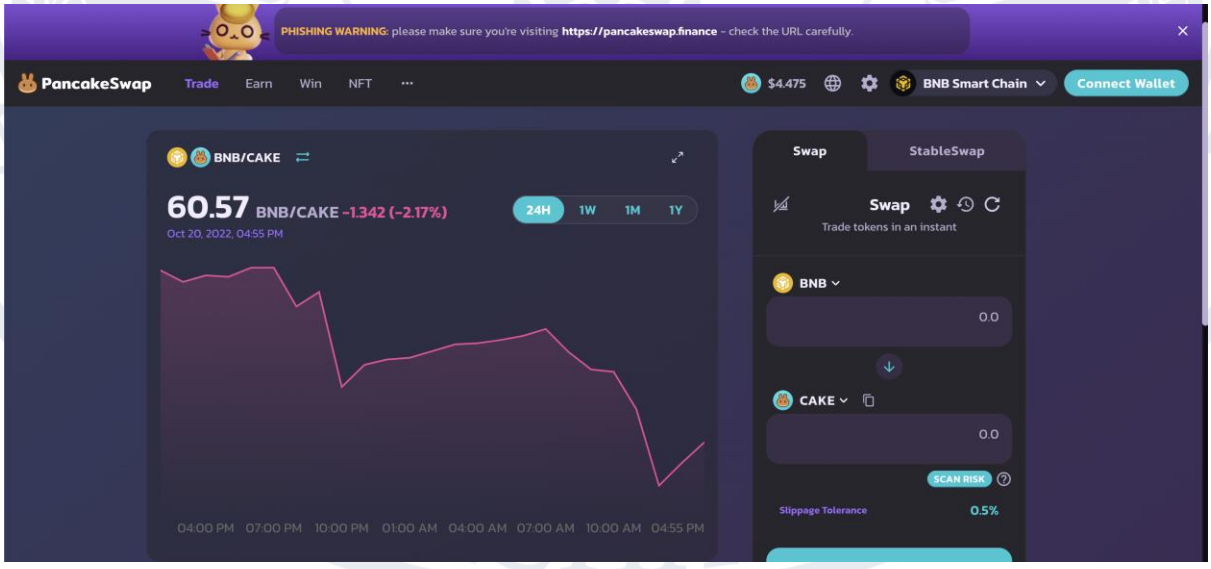


Рисунок 1.5 – огляд PancakeSwap

Дані сервіси є яскравим представником реалізації підходу web 3.0, на яких швидко можна обміняти певні токени, також можна зробити певний депозит, де кошту заморожуються, і інтервально відбувається начислення відсотків. Сервісі є дуже подібними, але працюють у різних мережах блокчейнів, PancakeSwap – у Binance Smart Chain, а Uniswap, відповідно, у Ethereum Smart Chain.

1.2.6 DeSo / BitClout

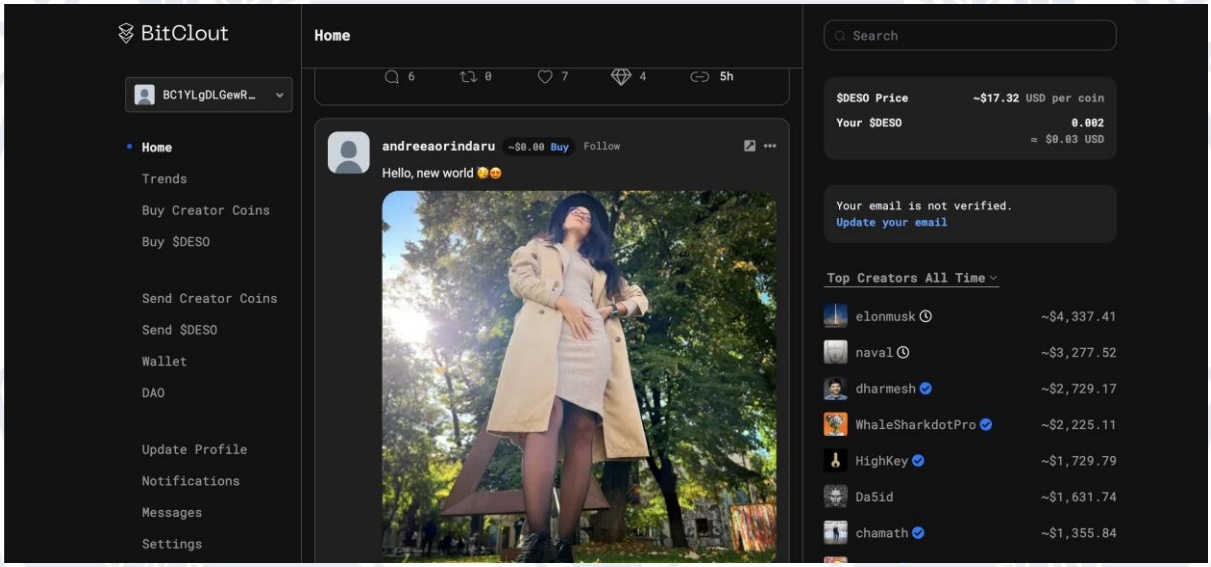


Рисунок 1.6 – огляд BitClout

BitClout є дочірнім проектом DeSo, що розшифровується, як децентралізований соціальний блокчейн. Працює на власному блокчейні, концепція схожа до Twitter, проте кожен користувач має певну вартість.

1.3 Смарт-контракт

Смарт-контракти - це просто програми, що зберігаються на блокчейні і запускаються при виконанні заздалегідь визначених умов. Зазвичай вони використовуються для автоматизації виконання угоди, щоб всі учасники могли бути відразу впевнені в результаті, без участі посередників або втрати часу. Вони також можуть автоматизувати робочий процес, запускаючи наступну дію при виконанні певних умов.

Смарт-контракти працюють за допомогою простих операторів "якщо/коли...то...", які записуються в код на блокчейні. Мережа комп'ютерів виконує дії, коли заздалегідь визначені умови були виконані та перевірені. Ці дії можуть включати переказ коштів відповідним сторонам, реєстрацію транспортного засобу, надсилання повідомлень або видачу квитка. Після завершення транзакції блокчейн оновлюється. Це означає, що транзакція не може бути змінена, і тільки сторони, яким було надано дозвіл, можуть бачити результати.

У смарт-контракті може бути стільки умов, скільки необхідно для того, щоб задовольнити учасників, що завдання буде виконано задовільно. Для встановлення умов учасники повинні визначити, як транзакції та їх дані будуть представлені в блокчейні, домовитися про правила "якщо/коли...то...", які регулюють ці транзакції, дослідити всі можливі винятки та визначити рамки для вирішення спорів.

Потім смарт-контракт може бути запрограмований розробником - хоча все частіше організації, які використовують блокчейн для бізнесу, надають шаблони, веб-інтерфейси та інші онлайн-інструменти для спрощення структурування смарт-контрактів.

1.4 Спортивне програмування

Спортивне програмування – це такий підвид програмування, коли люди змагаються між собою у вирішення алгоритмічних задач. Основні критерії при підведенні результатів:

- Час витрачений на розв’язок задачі
- Складність алгоритму і відповідно час роботи алгоритму при певних даних
- Оперативна пам’ять витрачена алгоритмом під час роботи
-

Спортивне програмування це дуже цікавий і ефективний спосіб підняти свій рівень, та цікаво провести час. Але для того щоб показувати високі результати потрібні постійні тренування та вивчення/покращення знань нових алгоритмів та структур даних.

На більшості змагань визначення результатів проводиться автоматично за допомогою спеціальних систем. Кожен розв'язок завдання запускається на сервері. На вхід цьому розв'язку подається список тестових прикладів (зазвичай секретний). У більшості випадків вирішення проблем маркуються за принципом «все або нічого», тобто якщо вирішення спрацювало неправильно на хоча б одному з тестових прикладів, воно не зараховується. Однак, деякі змагання використовують процентну систему оцінювання, тобто за розв'язок дають стільки відсотків, скільки відсотків тестових прикладів було розв'язано правильно.

Створення автоматизованої системи для розв'язування таких завдань і є одною з основних цілей роботи.

1.5 Play2Earn, Move2Earn, Learn2Earn

При створення додатку з концепцією Learn2Earn (Навчайся щоб заробити) було отримано натхнення з подібний концепцій, де для заробітку потрібно, чи грати в ігри, чи рухатись, чи ще щось подібне.

Play2Earn, грайте, щоб заробити, або P2E — це новий спосіб, натхненний блокчейном, заробляти внутрішньоігрові активи та винагороди, граючи у ваші улюблені віртуальні та відеоігри. Оскільки блокчейн і метавсесвіт підтримують функцію «грай, щоб заробити», вони децентралізовані, безпечні та надійні.

Це платформа, яка дозволяє користувачам накопичувати величезні NFT (незамінні токени) і криптовалюти, граючи в ігри. NFT — це новий тип токенів, який набирає популярності завдяки своїм унікальним властивостям. Вони отримують винагороду для геймерів P2E і не підпадають під ті ж правила, що й традиційні криптовалюти. Вони можуть представляти цінні цифрові активи та предмети колекціонування, які можна обміняти на реальні гроші.

Play2Earn — цікава нова концепція, яка поєднує в собі ігри та заробіток. Ви можете розглядати це як децентралізований ринок, де ігрові активи можна торгувати за реальну вартість. Хоча це все ще на ранніх стадіях, грай, щоб заробити, багатообіцяюча і може стати ключовим гравцем у майбутньому метавсесвіту.

На основі play2earn концепції з’явилась move2earn, тобто рухайся, для того щоб отримати винагороду.

Наразі більшість ігор Move2Earn – це або ходьба, біг, або додаток Fitbit, який відстежує ваші тренування, і ви отримуєте винагороду жетонами. Користувачі отримують NFT, і чим більше у вас NFT певного рівня, ви зможете заробити більше токена, і цей токен потім можна буде продати на публічній біржі, або користувачі зможуть обміняти ці токени на інші речі.

Ще одна нова сфера — впровадження протоколу Move2Earn, щоб заохотити людей використовувати більш стійкі або екологічні варіанти мобільності. Користувачі можуть зменшити викиди вуглецю, які можна конвертувати в NFT, криптотокени чи інші винагороди. Гравці мобільності можуть отримати вуглецеві кредити за індивідуальну поведінку та отримати додатковий дохід. Ідея полягає в гейміфікації всієї подорожі для водіїв і винагороді користувачів за їх стійку поведінку.

Learn2earn – відповідно аналогічний підхід, де для того щоб отримати винагороду потрібні не фізичні старанні, а розумові. В даному випадку це розв’язування задач спортивного програмування.

1.6 Визначення децентралізованого додатку

Децентралізовані додатки (dApps) - це цифрові додатки або програми, які існують і працюють на блокчейні або одноранговій (P2P) мережі комп'ютерів, а не на одному комп'ютері. DApps (також звані "dapps") знаходяться поза межами компетенції та контролю одного органу влади. DApps, які часто будуються на платформі Ethereum, можуть бути розроблені для різних цілей, включаючи ігри, фінанси та соціальні мережі.

- Децентралізовані додатки - також відомі як "dApps" або "dapps" - це цифрові додатки, які працюють на мережі комп'ютерів блокчейн замість того, щоб покладатися на один комп'ютер.
- Оскільки dApps є децентралізованими, вони вільні від контролю та втручання єдиного органу керування.
- Серед переваг dApps - захист конфіденційності користувачів, відсутність цензури та гнучкість розробки.
- До недоліків можна віднести потенційну неможливість масштабування, проблеми з розробкою користувацького інтерфейсу та труднощі з внесенням змін до коду.

Стандартний веб-додаток, такий як Uber або Twitter, працює на комп'ютерній системі, яка належить і експлуатується організацією, що дає їй повну владу над додатком і його роботою. З одного боку може бути декілька користувачів, але внутрішня частина контролюється однією організацією.

DApps можуть працювати в мережі P2P або в мережі блокчейн. Наприклад, BitTorrent, Tor і Popcorn Time - це додатки, які працюють на комп'ютерах, що є частиною P2P-мережі, в якій кілька учасників споживають контент, подають або роздають контент або одночасно виконують обидві функції.

У контексті криптовалют dApps працюють на мережі блокчейн у публічному, децентралізованому середовищі з відкритим вихідним кодом і є вільними від контролю та втручання з боку будь-якого окремого органу влади. Наприклад, розробник може створити додаток, подібний до Twitter, і розмістити його на блокчейні, де будь-який користувач може публікувати повідомлення. Після публікації ніхто, включаючи творців додатку, не може видалити повідомлення.

Отоже Etherium Dapp - це децентралізовані додатки, які працюють і розробляються з використанням платформи Ethereum. Ethereum dApps використовують смарт-контракти для своєї логіки. Вони розгортаються в мережі Ethereum і використовують блокчейн платформи для зберігання даних. Децентралізований додаток (також відомий як dApp або dapp) працює на блокчейні або одноранговій мережі комп'ютерів. Це дозволяє користувачам здійснювати транзакції безпосередньо один з одним, а не покладатися на центральний орган. Користувач dApp сплачує розробнику певну суму криптовалюти за завантаження та використання вихідного коду програми. Вихідний код відомий як смарт-контракт, який дозволяє користувачам здійснювати транзакції без розкриття особистої інформації.

Також централізовані додатку в свою чергу такі як Twitter, Facebook, Instagram та Netflix. Банки та інші фінансові установи використовують централізовані додатки для надання своїм клієнтам доступу до їхніх рахунків в режимі онлайн.

Прикладом децентралізованого додатку є Peepeth, соціальна мережа, альтернативна Twitter. Cryptokitties - dApp-гра, яка дозволяє користувачам купувати та продавати віртуальних котів. MakerDAO - децентралізований кредитний сервіс, що підтримує стейблкоїн Dai і дозволяє користувачам відкривати забезпечену боргову позицію (CDP).

РОЗДІЛ 2 ПРОЕКТНА ЧАСТИНА

2.1 Вибір інструментарію для розробки

Для реалізації сервісу потрібно обрати стек технологій, який буде гнучким, досить швидким, там надавати можливість легкого масштабування. Отже розпочнем з головного, з того як буде відбуватись взаємодія з користувачі з сервісом.

2.1.1 Solidity

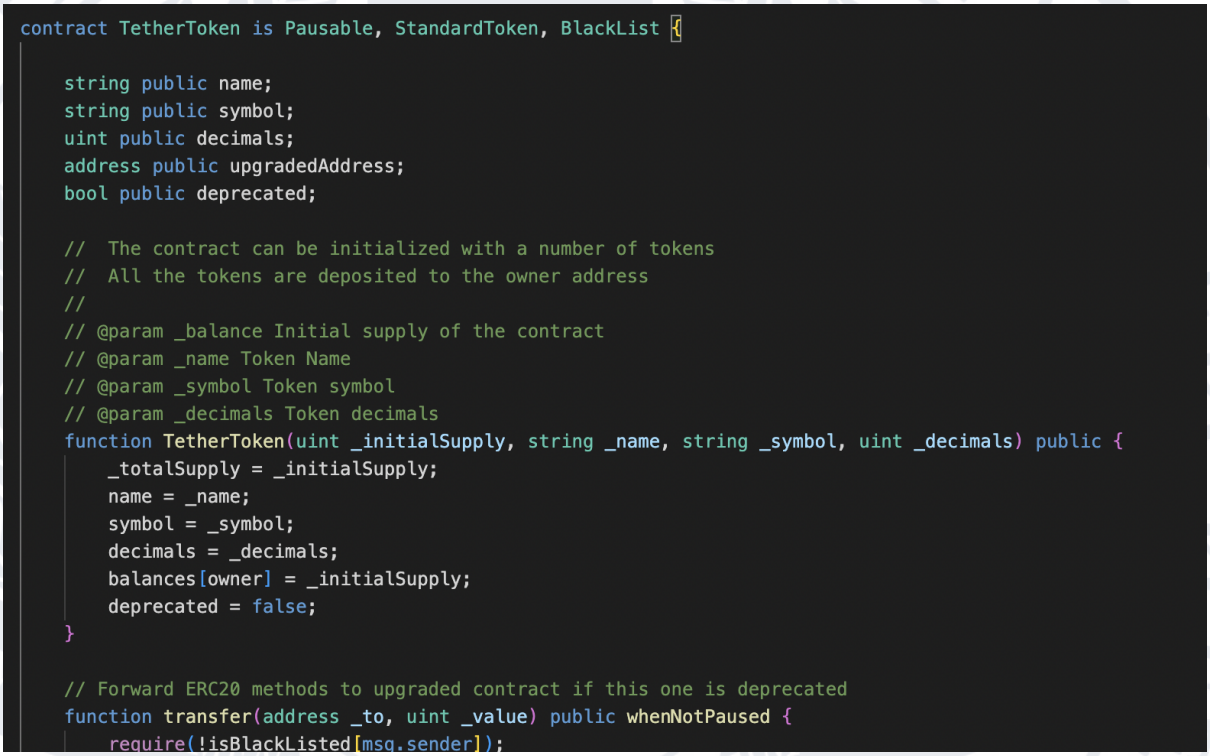
Solidity — об'єктно-орієнтована та предметно-орієнтована мова програмування розумних контрактів для платформи Ethereum.

Мова була запропонована в серпні 2014 року Гейвіном Вудом (Gavin Wood). Надалі розробка мови була виконана під керівництвом Крістіана Райтвизнера (Christian Reitwiessner) командою Solidity в рамках проєкту Ethereum. Це одна з чотирьох мов (три інші: Serpent, LLL і Mutan), спроєктованих для трансляції в байт-код віртуальної машини Ethereum. Отримала широке поширення з появою технологій блокчейну, зокрема стека технологій на основі Ethereum, для створення програмного забезпечення розумних контрактів.

Solidity — статично типізована JavaScript-подібна мова програмування, створена для розробки розумних контрактів, які працюють на віртуальній машині Ethereum (EVM). Програми на мові Solidity транслюються в байткод EVM. Solidity дозволяє розробникам створювати самодостатні програми, що містять бізнес-логіку, результуючу в транзакційні записи блокчейну.

Використання синтаксису ECMAScript за задумом Вуда має допомогти прийняттю мови дійсними веброзробниками. Однак, на відміну від ECMAScript, мова отримала статичну типізацію змінних і динамічні типи значень. Порівняно з компільованими в такий же байт-код мовами Serpent і Mutan, мова Solidity має важливі відмінності. Підтримуються комплексні змінні контрактів, включаючи довільні ієрархічні відображення

(mappings) і структури. Контракти підтримують спадкування, включаючи множинне і C3-лінеаризацію. Підтримується бінарний інтерфейс програмування (ABI), що має безліч типобезпечних функцій в кожному контракті (пізніше з'явився також і у Serpent). Специфікована система документування коду, для пояснення послідовності викликів, що отримала назву «Специфікації природною мовою Ethereum» (Ethereum Natural Format Specification).



```
contract TetherToken is Pausable, StandardToken, BlackList {

    string public name;
    string public symbol;
    uint public decimals;
    address public upgradedAddress;
    bool public deprecated;

    //  The contract can be initialized with a number of tokens
    //  All the tokens are deposited to the owner address
    //
    // @param _balance Initial supply of the contract
    // @param _name Token Name
    // @param _symbol Token symbol
    // @param _decimals Token decimals
    function TetherToken(uint _initialSupply, string _name, string _symbol, uint _decimals) public {
        _totalSupply = _initialSupply;
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        balances[owner] = _initialSupply;
        deprecated = false;
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function transfer(address _to, uint _value) public whenNotPaused {
        require(!isBlackListed[msg.sender]);
```

Рисунок 2.1 – приклад коду на Solidity

2.1.2 Ganache

Досить зручний програмний продукт, котрий дає можливість розвернути локально Ефіріум блокчейн, з допомгою якого можна надалі вести розробку і тестування, як і web 3.0 інтеграцій, так і смарт контрактів.

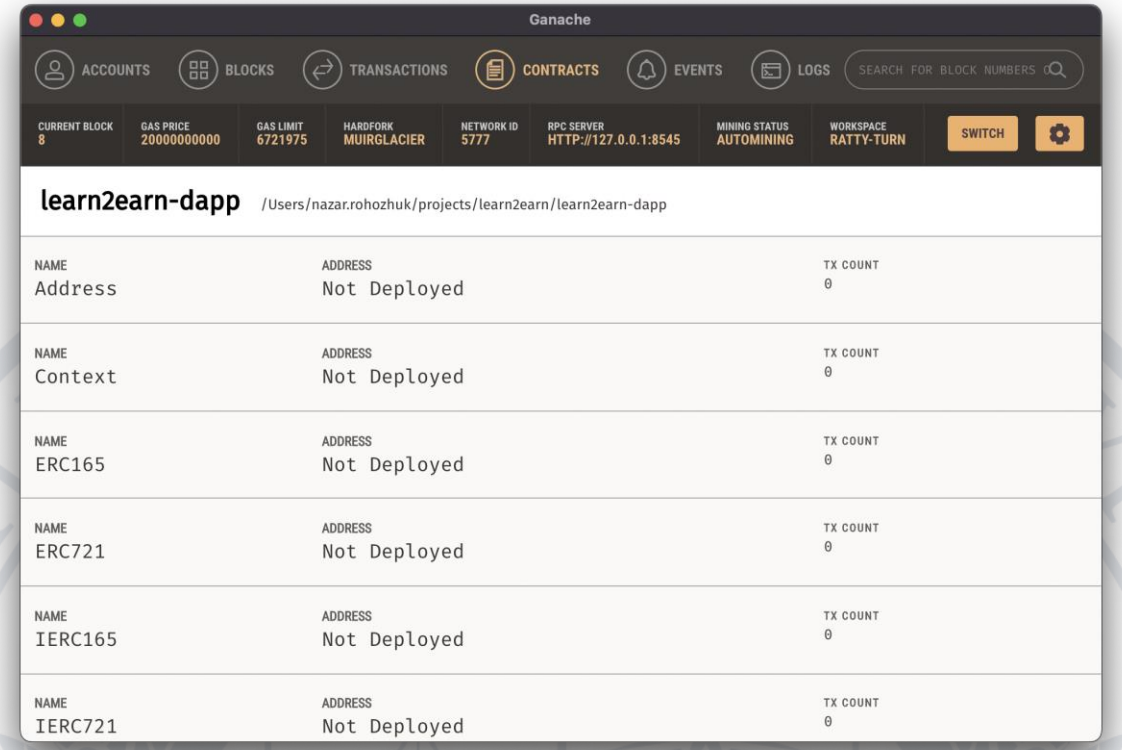


Рисунок 2.2 – огляд Ganache

2.1.3 JavaScript

JavaScript - обєктно-прототипна мова програмування, що здебільшого використовується у веб-розробці. До особливостей можна віднести те, що завдяки бібліотекам та фреймворкам можна реалізувати будь яку архітектуру, завдяки чому мова досить універсальна. Використовується як єдина можливість писати браузерні додатки. Має безліч бібліотек та фреймворків для реалізації майже будь чого. Мова JavaScript використовується для:

написання сценаріїв вебсторінок для надання їм інтерактивності

створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js)

- програмування на боці сервера (Node.js(Express.js))
- стаціонарних застосунків (Electron, NW.js)

- мобільних застосунків (React Native, Cordova)
- сценарії в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter)
- всередині PDF-документів тощо.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. В перші роки існування, більшість професійних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-аматорів. Поява AJAX змінила ситуацію та звернула увагу професійної спільноти до мови, а її подальші модифікації за стандартами ES6+ внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

Оскільки JavaScript є інтерпретованою мовою програмування, без строгої типізації, і може виконуватися в різних середовищах, кожне зі своїми власними особливостями сумісності, програміст має бути уважним, і повинен перевіряти, що його код виконується як очікується в широкому переліку можливих конфігурацій. Типізація вважається одною з ключових проблем JavaScript, тому восени 2012 року, компанія Microsoft презентувала мову програмування TypeScript, що компілюється в JavaScript та містить декілька важливих для програмістів доповнень, що полегшують розробку.

2.1.4 NodeJS

Node.js - це кросплатформенне середовище виконання JavaScript з відкритим вихідним кодом та бібліотека для запуску веб-додатків за межами браузера клієнта. Райан Даль розробив її у 2009 році. Розробники використовують Node.js для створення веб-додатків на стороні сервера, і він

ідеально підходить для додатків з інтенсивним використанням даних, оскільки використовує асинхронну модель, керовану подіями.

Існує багато причин, чому ввжається краще використовувати NodeJs для серверної частини нашого додатку, деякі з них розглянуті нижче:

- NodeJs побудований на движку V8 Google Chrome, і з цієї причини його час виконання дуже швидкий і він працює дуже швидко.
- У менеджері пакетів NodeJS доступні понад 50 000 пакетів, тому розробники можуть у будь-який час імпортувати будь-який з них відповідно до необхідної функціональності, що значно економить час.
- Оскільки NodeJS не потрібно чекати, поки API поверне дані, тому для побудови веб-додатків, що працюють в режимі реального часу та інтенсивно використовують дані, він є дуже корисним. –
- Він повністю асинхронний за своєю природою, що означає, що він повністю не блокується.
- Час завантаження аудіо або відео скорочується завдяки NodeJs, оскільки відбувається краща синхронізація коду між клієнтом і сервером через наявність однакової кодової бази.
- Оскільки NodeJs має відкритий вихідний код і є нічим іншим, як фреймворком JavaScript, то для розробників, які вже звикли до JavaScript, почати розробляти свої проекти на NodeJs дуже легко.

2.1.5 NestJS

NestJS – потужний фреймворк для побудовий додатків з використанням платформи Node.JS, при створенні автори надихались екосистемою Sping з всесвіту Java та front-end фреймворком Angular.

Має велику кількість вбудованих та адаптованих бібліотек, для роботи з Rest API, GraphQL, реляційними базами даних з допомгою бібліотеки typeorm та налаштувань над нею, роботу з не реляційними

базами, такими як MongoDB з допомогою бібліотеки typegoose, побудови мікросервісної архітектури, роботи з брокерами повідомлень.

Також використовує патерн dependency injection.

Dependency Injection (DI) - шаблон проєктування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію управління» (англ. Inversion of control, IoC) для розв'язання (отримання) залежностей. Впровадження — це передача залежності (тобто, сервісу) залежному об'єкту (тобто, клієнту). Передавати залежності клієнту замість дозволити клієнту створити сервіс є фундаментальною вимогою до цього шаблону проєктування.

У сервісі є багато вбудованих компонент, одна з базових це, так званий, модуль. Модуль - це клас з декоратором @Module (). Декоратор @Module () надає метадані, які Nest використовує для організації структури програми. Кожна програма Nest має як мінімум один модуль, корневий модуль. Корневий модуль - це місце, де Nest починає впорядковувати дерево додатків. Фактично, корневий модуль може бути єдиним модулем в вашому додатку, особливо коли додаток маленький, але це не має сенсу. У більшості випадків у вас буде кілька модулів, кожен з яких має тісно пов'язаний опцій. В Nest модулі за замовчуванням є Синглетон, тому ви можете без проблем використовувати один і той же екземпляр компонента між двома і більше модулями.

Модульна система Nest поставляється з функцією динамічних модулів. Це дозволяє створювати власні модулі без будь-яких зусиль.

В якості мови використовується TypeScript. Багато функціоналу реалізовується досить зручним способом через декоратори.

Розпочати роботу з nest проектом можна всього з 2 команд:

- npm i -g @nestjs/cli
- nest new demo-app

Після цього буде згенерований кістяк сервісу.

В практичній частині магістерської роботи використовується у основному сервісі обробки запитів та комунікації з блокчейном, запуску коду завдання та перевірка на тестових даних.

```
import { Controller, Get, Query, Post, Body, Put, Param, Delete } from '@nestjs/common';
import { CreateCatDto, UpdateCatDto, ListAllEntities } from './dto';

@Controller('cats')
export class CatsController {
  @Post()
  create(@Body() createCatDto: CreateCatDto) {
    return 'This action adds a new cat';
  }

  @Get()
  findAll(@Query() query: ListAllEntities) {
    return `This action returns all cats (limit: ${query.limit} items)`;
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return `This action returns a #${id} cat`;
  }

  @Put(':id')
  update(@Param('id') id: string, @Body() updateCatDto: UpdateCatDto) {
    return `This action updates a #${id} cat`;
  }
```

Рисунок 2.3 Приклад контролера на nest.js

2.1.6 websocket

WebSocket — це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком.

З допомогою WebSocket можна будувати real-time веб додатки, використовується у меседжерах, онлайн чатах, біржах.

В практичній частині магістерської роботи використовується, наприклад, як канал для відображення статусу перевірки здачі задачі.

2.1.7 typeorm

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm"

@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id: number

    @Column()
    firstName: string

    @Column()
    lastName: string

    @Column()
    age: number
}
```

Рисунок 2.4 – приклад використання typeorm

Typeorm – потужна ORM для роботи з реляційними базами даних, написана мовою TypeScript та використовує багато його можливостей, у вигляді рефлексії та декораторів. Має дуже потужну систему автоматичних міграцій. В NestJS, який використовується в практичній частині бакалаврської роботи, присутні адаптери для typeorm для того щоб зробити роботу максимально зручною.

В практичній частині бакалаврської роботи використовується для комунікації з PostgreSQL.

2.1.8 class-validator, class-transformer

```
export class Post {
  @MinLength(10)
  @MaxLength(20)
  title: string;

  @Contains('hello')
  text: string;

  @IsInt()
  rating: number;

  @IsEmail()
  email: string;

  @IsFQDN()
  site: string;

  @IsDate()
  createDate: Date;

  @ArrayNotEmpty()
  @ArrayMinSize(2)
  @ArrayMaxSize(5)
  @MinLength(3, { each: true, message: 'Tag is too short. Minimal length is $valu
  @MaxLength(50, { each: true, message: 'Tag is too long. Maximal length is $valu
  tags: string[];
```

Рисунок 2.5 – приклад використання class-validator

class-validator, class-transformer – бібліотеки для зручної роботи та валідації даних та автоматичного створення об’єктів базуючись на схемах. Генерація схем здійснюється автоматично, базуючись на декораторах.

2.1.9 React JS

React — це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду — “компонентів”.

З самого початку React був спроектований так, щоб його можна було впроваджувати поступово. Тобто ви можете додавати так мало або так багато React-у, як вам потрібно.

Досить швидкий, так як використовується таке поняття, як shadow dom, тобто перед тим як рендерити об’єкти безпосередньо в браузері, спочатку рендеряться в віртуальний dom, і потім тільки зміни застосовуються до справжнього dom.

Використовується так званий формат JSX, це такий підхід коли html розмітку можна вставляти в один файл з js кодом.

У практичній частині магістерської роботи застосовується для написання клієнтської частини, реалізації з web 3.0.

```
const useDocumentTitle = title => {
  useEffect(
    () => {
      document.title = title;
    },
    [title]
  );
};
function Counter() {
  const [count, setCount] = useState(0);
  const increment = () => setCount(count + 1);
  const title = `You clicked ${count} times.`;
  useDocumentTitle(title);
  return (
    <div>
      <h3>{count}</h3>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Рисунок 2.4 – приклад коду з використанням React.

2.1.10 Docker

Docker - одна з найпопулярніших контейнерних платформ, що привертає увагу багатьох команд розробників. Все більше компаній переходять на Docker завдяки його надійності, продуктивності та функціональності.

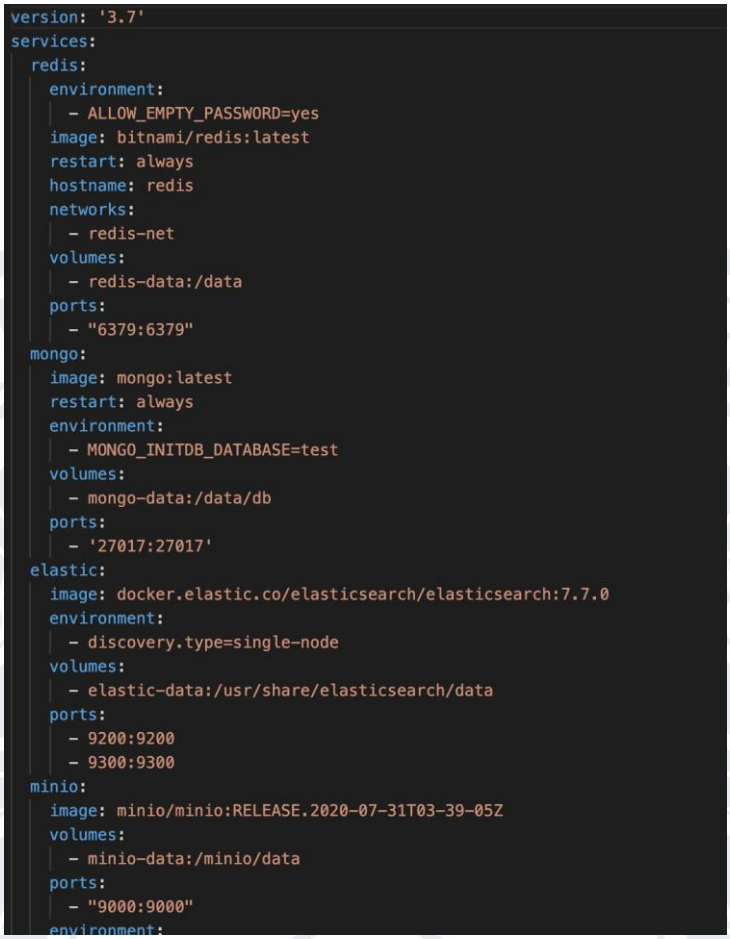
В основному використовується як платформа розробки програмного забезпечення для розробки розподілених додатків, які ефективно працюють

в різних середовищах. Завдяки тому, що програмна система є агностичною, розробникам не потрібно турбуватися про проблеми сумісності. Упаковка додатків в ізольовані середовища (контейнери) також полегшує розробку, розгортання, підтримку та використання додатків.

Оскільки Docker використовує віртуалізацію для створення контейнерів для зберігання додатків, концепція може здатися схожою на віртуальні машини. Хоча обидва представляють собою ізольовані віртуальні середовища, що використовуються для розробки програмного забезпечення, між контейнерами і віртуальними машинами існують важливі відмінності. Найважливіша відмінність полягає в тому, що контейнери Docker легші, швидші та більш ресурсоефективні, ніж віртуальні машини.

2.1.11 docker-compose

docker-compose – утиліта для зручного декларативного розвертання docker контейнерів. Принцип роботи досить простий, на вхід приймається маніфест docker-compose.yaml де описано які контейнери з якими образами потрібно запустити, та яка мережева взаємодія повинна бути налаштована, і утиліта створює потрібні ресурси та контейнери.



```yaml
version: '3.7'
services:
  redis:
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
    image: bitnami/redis:latest
    restart: always
    hostname: redis
    networks:
      - redis-net
    volumes:
      - redis-data:/data
    ports:
      - "6379:6379"
  mongo:
    image: mongo:latest
    restart: always
    environment:
      - MONGO_INITDB_DATABASE=test
    volumes:
      - mongo-data:/data/db
    ports:
      - '27017:27017'
  elastic:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.7.0
    environment:
      - discovery.type=single-node
    volumes:
      - elastic-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300
  minio:
    image: minio/minio:RELEASE.2020-07-31T03-39-05Z
    volumes:
      - minio-data:/minio/data
    ports:
      - "9000:9000"
    environment:
```

Рисунок 2.5 – приклад docker-compose маніфесту

2.1.12 Мікросервісна архітектура, брокери повідомлень, масштабування

Для початку слід визначити основні поняття, що стосуються мікросервісної архітектури

Брокер повідомлень (або диспетчер черги) - це програмний застосунок, який приймає і віддає сполучення між окремими модулями / додатками всередині деякої складної системи, де модулі / додатки повинні спілкуватися між собою - тобто пересилати дані один одному.

Розподілена система - така система, яка працює відразу на безлічі машин, що утворюють цілісний кластер. Кластер це набір комп'ютерів /

серверів, об'єднаних мережею і взаємодіючих між собою. Найважливіші плюси такого підходу - високодоступних і відмовостійкість.

Гарантія доставка – підхід коли повідомлення попадає до брокера, і не зникає з черги доки сервіс не відповість, що повідомлення доставлене та оброблене.

Вертикальна масштабованість - це нарощування ресурсів, тобто збільшення кількості ядер, оперативної пам'яті і т.д. на одному сервері.

Переваги:

- досить просто і зрозуміло використовувати

Недоліки:

- не можна нарощувати нескінченно.
- при додаванні ресурсів (оперативна пам'ять і т.д.) зазвичай доводиться вимикати сервер, що погано впливає на відмовостійкість.

Горизонтальна масштабованість - це додавання нових серверів з будь-якими характеристиками в обчислювальний кластер.

Переваги:

- Немає таких проблем, як у вертикальній масштабованості

Недоліки:

- Не скрізь підтримується горизонтальна масштабованість
- Не всі системи працюють в кластерах, а ті, які в них працюють, зазвичай досить складні в експлуатації

Наприклад більшість баз даних неможливо або досить важко налаштувати під роботу у кластерному режимі.

Далі наведено основні брокери повідомлень, визначимо переваги та недоліки, та оберемо, який буде використовуватись у сервісі обробки відео.

Apache Kafka - розподілений горизонтально масштабований відмовостійкий програмний брокер повідомлень.

Додатки (генератори) відправляють повідомлення (записи) на вузол Kafka (брокер), і повідомлення про прийняття обробляються іншими додатками, так званими споживачами. Зазначені повідомлення зберігаються, а споживачі підписуються для отримання нових повідомлень. Kafka гарантує, що всі повідомлення будуть упорядковані саме в тій послідовності, в якій надійшли. Kafka не відслідковує, які записи зчитуються споживачем і після цього видаляються, а просто зберігає їх протягом заданого періоду часу. Споживачі самі опитують Kafka, чи не з'явилося у нього нових повідомлень, і вказують, які записи їм потрібно прочитати.

Цей підхід схожий на бібліотеку з читальним залом, коли хто завгодно, бере книгу (повідомлення), читає її в читальному залі, потім віддає назад. А книги, коли застарівають, просто викидаються з бібліотеки.

RabbitMQ, як і Kafka, теж розподілений горизонтально масштабований відмовостійкий програмний брокер повідомлень.

Додатки (publishers) надсилають повідомлення на вузол RabbitMQ (exchange), при цьому RabbitMQ відсилає назад додаткам підтвердження, що повідомлення отримано. Одержувачі (consumers) завжди пов'язані з RabbitMQ по TCP і чекають, коли RabbitMQ проштовхне (push) їм повідомлення. Одержувачі підтверджують отримання повідомлення або повідомляють про невдачу. Якщо доставка невдала, то RabbitMQ

проштовхує повідомлення до тих пір, поки воно не буде доставлене. Після успішної доставки повідомлення видаляється з черги.

Цей підхід можна порівняти з поштовим відділенням і листоношою, коли посилки (повідомлення) приходять від відправників на пошту, звідти розсилаються по поштовим відділенням, а потім листоноша розносить їх по адресах і переконується, що посилка дійшла до одержувача.

NATS – відносно молодий проект, написаний на мові Go. По замовчуванню має лише можливість Pub-Sub (без гарантії доставки, без черги повідомлень). Проте є налаштування та модулі які додають черги (NATS Streaming, STAN). Та найпотужніше над налаштування Jet Stream, яке включає в себе вище перераховані можливості.

Отже проаналізувавши доступні AMQP вибір впав на RabbitMQ з наступних причин

- Production ready рішення (на відміну від NATS)
- Перевірене та протестоване на великій кількості проектів
- Просте у використання (на відміну від Apache Kafka)
- Багато можливостей на налаштувань з коробки

2.1.13 PostgreSQL

PostgreSQL - це популярна вільна об'єктно-реляційна система управління базами даних. PostgreSQL базується на мові SQL і підтримує численні можливості.

Переваги:

- Підтримка БД необмеженого розміру
- Потужні та надійні механізми транзакцій та реплікацій
- Розширювана система вбудованих мов програмування і підтримка завантаження С-сумісних модулів

- Велика кількість унікальних функцій та алгоритмів для побудови індексів та запитів

Порівняно з іншими проєктами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

PostgreSQL підтримує одночасну модифікацію БД декількома користувачами за допомогою механізму Multiversion Concurrency Control (MVCC). Завдяки цьому виконуються вимоги ACID, і практично відпадає потреба в блокуванні зчитування.

2.1.14 Redis

Redis – високопродуктивна key-value база даних, яка в якості сховище використовує оперативну пам’ять, доступ до яких здійснюється за ключем, також Redis можна використовувати як брокер повідомлень. Також є можливість з певним інтервалом записувати дані диск, для того щоб забезпечити надійність при раптових перезапусках. Основна перевага цією БД – швидкість роботи.

Зазвичай використовується для кешу даних, примітивного спілкування між сервісами, коли не потрібна гарантія доставки.

Також в Redis є багато алгоритмів та структур даних для зберігання різних структур. Має велику кількість мов які підтримують взаємодію з БД.

В сервісу обробки відео буде використовуватись саме для взаємодії між екземплярами одного контейнера, при роботі з WebSocket.

Основні типи даних Redis:

- Стрічки
- Списки
- Множини
- Хеш таблиці

- Впорядковані множини

2.1.15 The Graph

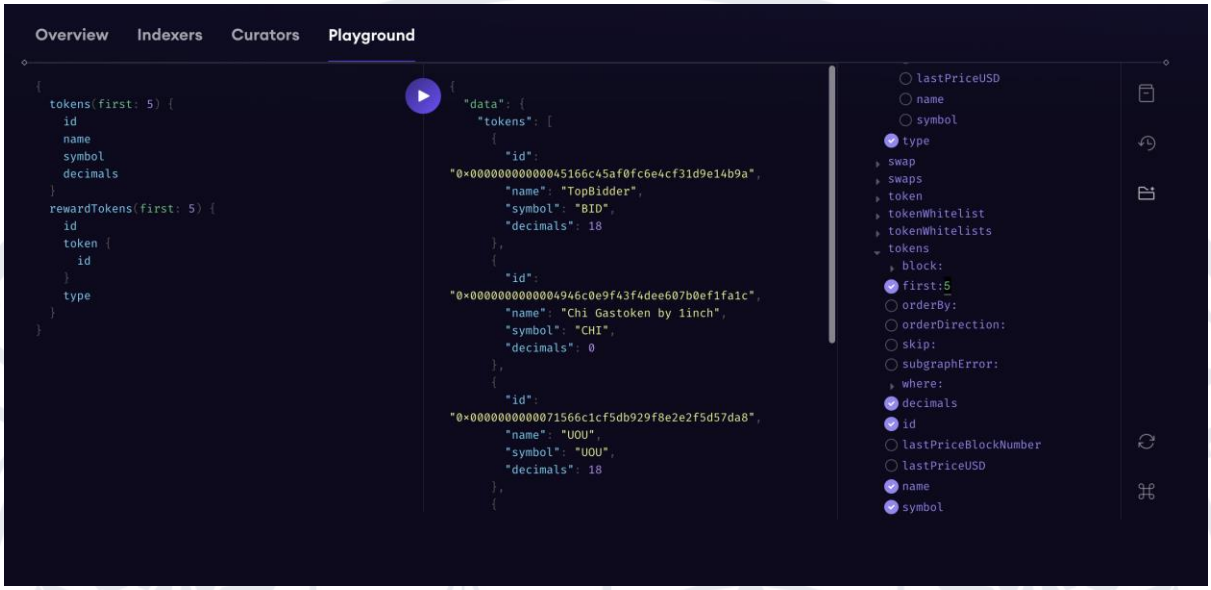


Рисунок 2.6 – приклад використання графу

Одним із поширених методів полегшення пошуку інформації є індексування. Суть полягає у використанні посилань, що вказують на конкретні записи даних, щоб отримати доступ до цих записів без необхідності пошуку в усій базі даних. Насправді цей підхід виник трохи раніше цифрової ери, і його можна простежити до фізичних книг, де індекси використовуються для позначення сторінок, на яких можна знайти певні записи (наприклад, розділи книг, записи енциклопедії тощо). .

Graph намагається використовувати цей підхід у контексті Web3. Це протокол індексування для запитів мереж блокчейну, таких як Ethereum і IPFS (InterPlanetary File System). Протокол дозволяє людям створювати відкриті API, які називаються підграфами, які роблять дані блокчейна легко доступними.

По суті, Graph — це інфраструктурний протокол, метою якого є надання нової функціональності, в даному випадку — індексації, в існуючу

екосистему блокчейну. Таким чином, він відповідає основним принципам типової системи блокчейн, але також розроблений для виконання деяких спеціалізованих функцій. Щоб досягти цього, учасники мережі The Graph виконують спеціальні завдання, що стосуються їхніх ролей у мережі. Існує три основних типи учасників мережі.

Індексатори — це оператори вузлів мережі The Graph. Для того, щоб приєднатися до мережі, індексатори повинні поставити нативний токен The Grapph, GRT, і запустити вузол. Їх основною функцією є індексація відповідних підграфів і обробка запитів. За ці послуги індексатори отримують комісію за запити та винагороду за індексування. Плата за запити обробляється через нещодавно запущену систему мікротранзакцій під назвою Scalar, яка полегшує прямі платежі індексаторам через державні канали. Тим часом винагороди за індексацію генеруються через 3% річну інфляцію за протоколом.

Куратори - це учасники мережі, які сигналізують індексаторам, які підграфи повинні бути проіндексовані мережею. Це працює так, що вони вносять GRT у зв’язувальну криву, щоб сигналізувати на певному підграфі та заробляти частину плати за запити для підграфів, на яких вони сигналізують.

Розробники DApp, споживачі даних та інші члени спільноти можуть бути кураторами та використовувати свої знання про простір Web3 для оцінки підграфів. Примітно, що винагорода за індексацію для певного підграфа залежить від кількості сигналів куратора, які він отримав, тобто чим популярнішим є підграф, тим вища винагорода за нього.

Делегатори — це люди, які можуть брати участь у мережі, делегуючи частки індексаторам. Таким чином, люди, які не хочуть керувати власними

вузлами, все одно можуть робити внесок у мережу та заробляти GRT, оскільки вони отримують частину комісії за запити та винагороди за індексування індексаторів, яким вони делегували. Делегатори повинні сплачувати комісію 0,5% кожного разу, коли вони делегують.

2.1.16 Ethers.js

Ethers.js — це бібліотека JavaScript, яка дозволяє розробникам взаємодіяти з блокчейном Ethereum. Бібліотека містить службові функції в JavaScript і TypeScript і має всі можливості гаманця Ethereum. Ethers.js зараз має версію 5.0.3. Ethers.js створено компанією Ethers і є відкритим кодом за ліцензією MIT.

Особливості:

- Зберігайте свої приватні ключі в клієнті, в цілості й схоронності
- Імпорт і експорт JSON гаманців (Geth, Parity і краудсейл)
- Імпорт і експорт мнемонічних фраз BIP 39 (12-слівні резервні фрази) і HD гаманців
- Мета-класи створюють JavaScript-об'єкти з будь-якого контрактного ABI, включаючи ABIv2 і Human-Readable ABI
- Можилвість підключатися до вузлів Ethereum по JSON-RPC, INFURA, Etherscan, Alchemy, Ankr або MetaMask
- Імена ENS (домени для адрес etherium)
- Модульність
- Велика документація

2.1.17 Remix project

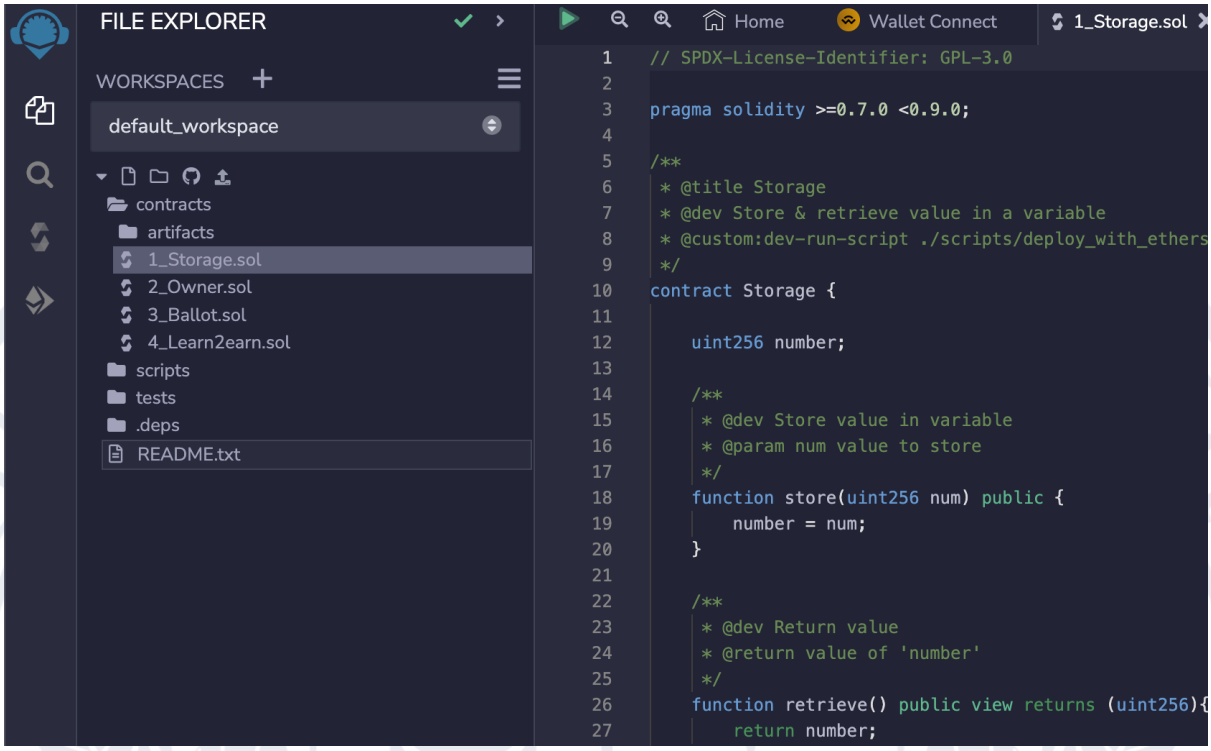


Рисунок 2.7 – remix ide

Remix Project - це платформа для розробки інструментів, що використовують архітектуру плагінів. Він складається з підпроектів, включаючи Remix Plugin Engine, Remix Libraries і, звичайно ж, Remix IDE.

Remix IDE - це веб- та десктопний додаток з відкритим вихідним кодом. Вона сприяє швидкому циклу розробки та має багатий набір плагінів з інтуїтивно зрозумілим графічним інтерфейсом. Remix використовується на всьому шляху контрактної розробки на мові Solidity, а також як майданчик для вивчення та викладання Ethereum.

РОЗДІЛ 3
ОПИС РОЗРОБКИ ТА ПРОЕКТУВАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

3.1 Загальна структура додатку

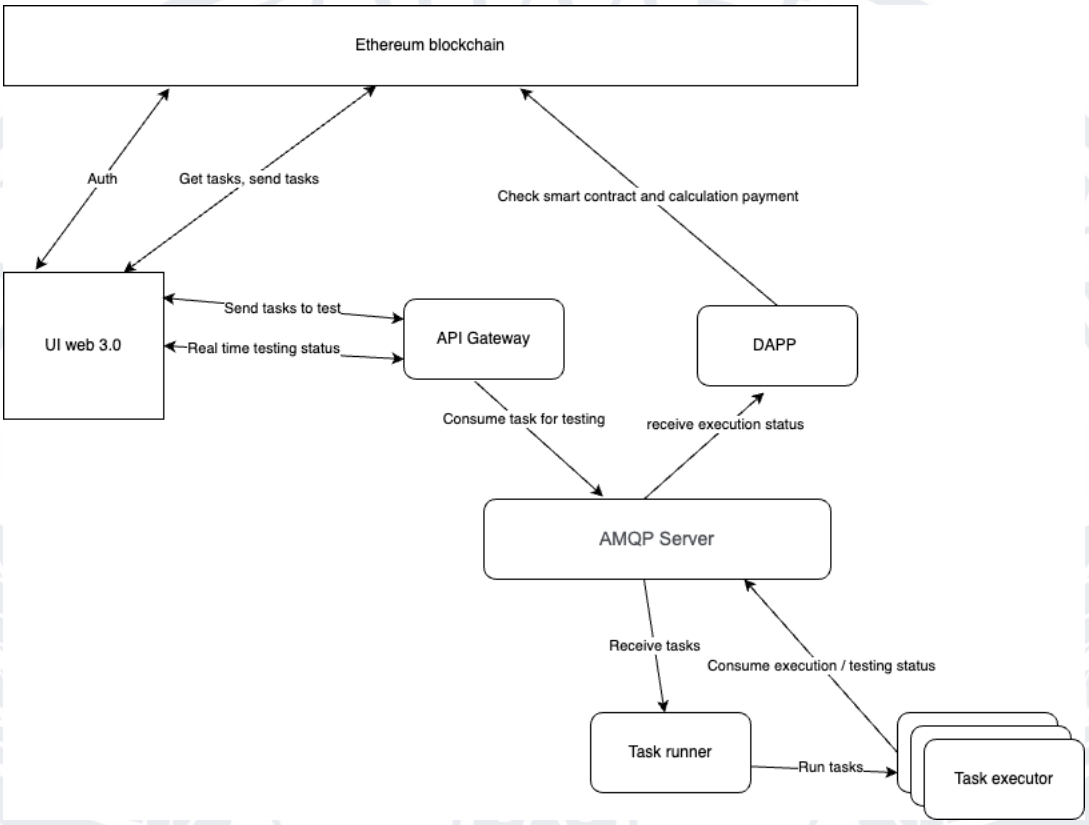


Рисунок 3.1 – структура додатку

При розробці сервісу було вирішено створити наступні сервіси:

API Gateway – основний сервіс, до якого приходять запити, та відбувається маршрутизація на інші сервіси.

DAPP – проект, який в собі містить всі міграції, та код смарт контрактів, які в подальшому потрапляють до блокчейну.

Task Runner – сервіс, який з брокера повідомлень отримує інформацію, про те які задачі потрібно відправити на перевірку.

Task Executor – сервіс який динамічно запускається на кожну задачу, і відзвітовує про статус виконання і перевірки рішення задачі.

Chain integration – сервіс, який відповідає за комунікацію з блокчейном, за роботу з смарт контрактом, також містить в собі логіку discovery токенів з задачами.

UI – відповідно веб додаток, який працює з REST API, WebSocket, і реалізовую web 3.0 з подальшою передачею інформації, через вказані протоколи.

3.2 Побудова частини додатку, яка відповідає за перевірку задач

Як було зазначено раніше створювати задачі та тест кейси зможе будь яка людина, за рахунок смарт контракту який буде завантажуватись до блокчейну, проте, всю цю інформацію також потрібно кешувати у себе. Для роботи з базою даних будемо використовувати TypeOrm.

Отже розпочнемо з побудови структури бази даних для зберігання задач, умов та тест кейсів.

3.2.1 Сутність умови задачі

```
@Entity()
export class Problem {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @Column()
  name: string;

  @Column()
  condition: string;

  @Column()
  checkerUrl: string;

  @Column()
  contractAddress: string;

  @ManyToOne( typeFunctionOrTarget: () => User)
  user: User;
}
```

Рисунок 3.2 – сутність задачі

Розберемо, яке поле за що відповідає:

- Id – ідентифікатор задачі у форматі uuid,
- Name – назва задачі
- Condition – умова задачі
- CheckerUrl – посилання на сирцівний файл – валідатор, який буде перевіряти задачу, файл зберігається на S3
- ContractAddress – посилання на контракт, в якому, власне, зберігається задача
- User – зв’язок на користувача, автора задачі

3.2.2 Сутність спроби на перевірку задачі

```
@Entity()
export class Attempt {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @OneToMany(
    typeFunctionOrTarget: () => AttemptResult,
    inverseSide: (result: AttemptResult) => result.attempt
  )
  results: AttemptResult[];

  @Column()
  hash: string;

  @Column( options: {
    type: 'enum',
    enum: Language,
  })
  language: Language;

  @Column()
  sourceUrl: string;

  @ManyToOne( typeFunctionOrTarget: () => Problem)
  problem: Problem;
}
```

Рисунок 3.3 – спроби на перевірку задачі

- Id – ідентифікатор задачі у форматі uuid,
- Results – результати роботи Checker
- Hash – хеш сирцівного коду, для подальшої перевірки на унікальність
- Language – мова програмування, якою написане рішення
- SourceUrl – посилання на сирцівний код, який зберігається на S3
- Problem – відношення на задачу

3.2.3 Сутність результату виконання

```
export enum ResultType {
  SUCCESS = 'SUCCESS',
  WRONG_ANSWER = 'WRONG_ANSWER',
  TIMEOUT = 'TIMEOUT',
  RUNTIME_ERROR = 'RUNTIME_ERROR',
  COMPILATION_ERROR = 'COMPILATION_ERROR',
}

@Entity()
export class AttemptResult {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @ManyToOne( typeFunctionOrTarget: () => Attempt,  inverseSide: (result: Attempt) => result.results)
  attempt: Attempt;

  @Column()
  testCase: number;

  @Column( options: {
    type: 'enum',
    enum: ResultType,
  })
  result: ResultType;
}
```

Рисунок 3.4 – сутність результату

- Id – ідентифікатор задачі у форматі uuid
- Attempt – відношення на спробу
- TestCase – номер тесту

- Result – результат виконання

Результат може приймати наступних значень:

- SUCCESS - успішно
- WRONG_ANSWER – неправильна відповідь
- TIMEOUT – програма виконувалась довше, за встановлений ліміт
- RUNTIME_ERROR – виконання програми завершилось помилкою, наприклад, відбулось звернення до неіснуючого елементу масиву
- COMPILATION_ERROR – під час компіляції програми відбулась помилка

3.2.4 Побудова сервісу, який безпосередньо буде займатись запуском перевірки задач

Так як в нас основний канал комунікації між мікро сервісами це AMQP, отже наш сервіс буде очікувати на нові спроби перевірки задачі, запускати перевірку, та звітувати назад у AMQP. Для початку визначимось з основним алгоритмом, та потрібними даними для перевірки.

1. Отримання інформації про спробу перевірити задачу
2. Завантажити сирцівний код з S3
3. Скомпілювати код, якщо це потрібно, і це не робилось раніше
4. Завантажити код валідатора
5. Скомпілювати код валідатора, якщо це не робилось раніше
6. Запустити одночасно рішення задачі та валідатор
7. Потоки валідатора Read спрямувати у потік додатку Write, а Write у Read
8. Засікти час, коли розпочалось тестування з цього моменту
9. Перевірити задачу
10. Якщо програма виконується довше ніж потрібно, вийти з неї, та з валідатора, та відправити відповідний статус.
11. Якщо все завершилось без помилок повернути статус SUCCESS

12.Пройтись по всім тест кейсам

13.Відзвітувати, що перевірки завершилась

Також слід визначити, що таке валідатор в нашому контексті. Для створення валідатору кінцевий користувач, повинен створити власний міні проект мовою TypeScript, використовуючи npm бібліотеку, в якій буде міститись інтерфейс, і власне, повинен реалізувати наступний інтерфейс.

```
export interface ResultChecker {
  isOk: boolean;
  comment?: string;
}

export interface Checker {
  getTotalCases(): Promise<number> | number;

  getInput(caseId: number): Generator<Promise<string> | string>;

  isCorrect(
    caseId: number,
    output: string[],
  ): Promise<ResultChecker> | ResultChecker;
}
```

Рисунок 3.5 – інтерфейс валідатору

Перший метод для реалізації, це getInput – метод який відповідає за потік даних, які будуть передаватись на вхід. Тут використано підхід генератору, для більш простої роботи з потоком. По суті самі ці дані будуть передаватись в input stream скомпільованого рішення. Далі як відпрацювала програма, буде зчитуватись все з output stream, і перевірятись з допомогою методу isCorrect.

Після розробки модулю тестування, потрібно розробити сервіс, який як і буде запускати ці тестування, так і обробляти результати тестування. Як

було сказано раніше, ми будемо використовувати AMQP. Отже створюємо контролер, який буде слухати потрібні нам події, і відповідно оновлювати дані у базі даних, та звітувати у websocket.

```
@Injectable()
export class TaskExecutionController {
  constructor(private readonly tasksService: TasksService) {}

  @MessagePattern('finish-test-case')
  async onFinishTestCase(
    @Payload() data: AttemptResultEventPayload,
    @Ctx() context: RmqContext,
  ) {
    await this.tasksService.addAttemptResult(data);
  }

  @MessagePattern('finish-checker')
  async onFinishChecker(
    @Payload() data: FinishCheckerEventPayload,
    @Ctx() context: RmqContext,
  ) {
    await this.tasksService.markAttempt(data, finished: true);
  }
}
```

Рисунок 3.6 – огляд контролеру виконання задач

Як видно тут все дуже просто, просто слухаємо події, та оновлюємо статус. Але це все так просто завдяки використанню nestjs та його модуля nestjs/microservices.

Далі пропоную оглянути роботу з базою даних, так як ми вже створили всі потрібні, так звані entity, можемо приступати безпосередньо до написання сервісу. TypeOrm підтримує два підходи

- Active Record – це коли кожна entity наслідується від базової entity, і відповідно отримує методи як і для створення, так і для оновлення, це досить зручно на невеликих проектах, і коли в нас прості дані, але це вважається антипатерном.

- Data Mapper – це коли є, так званий, репозиторій, який вміє працювати лише з Entity певного типу, і відповідно можна робити всі маніпуляції з Entity використовуючи цей репозиторій, зупинимо свій вибір на цьому підході.

Далі розглянемо декілька базових методів з сервісу, котрий працює з користувачами.

```
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User) private readonly userRepository: Repository<User>,
  ) {}

  async findAll(options?: FindManyOptions<User>): Promise<User[]> {
    return await this.userRepository.find(options);
  }

  async findOne(
    where: UserWhereUniqueInput | UserUnauthorizedWhere,
    options?: FindOneOptions<User>,
  ): Promise<User> {
    return await this.userRepository.findOne({
      where,
      ...options,
    });
  }
```

Рисунок 3.7 огляд роботи з базою даних

Як бачимо, робота з базою даних виглядає досить просто, за рахунок використання typeorm. Але також залишається в той же момент досить гнучкою. Також ознайомимось з методом create.

```
async create(data: CreateUser): Promise<User> {
  const exists = await this.findByEmail(data.email);
  if (exists) {
    throw new Error('User already exists');
  }
  const usersCount = await this.count();
  const user = this.userRepository.create({
    ...data,
    isActive: usersCount === 0,
  });
  return await this.userRepository.save(user);
}
```

Рисунок 3.8 огляд роботи з базою даних

3.2.5 Написання валідатора, огляд роботи додатку

Отже перейдемо до написання тестового валідатора, та перевірки роботи сервісу. Задача, яку буде перевіряти валідатор, звучить дуже просто. На вхід подається 2 числа N, M. На вихід потрібно вивести суму цих чисел.

Отже ми експортуємо нашу бібліотеку, і реалізовуємо з неї інтерфейс Checker, основна перевага в використанні даного підходу, це те що, написання подібних класів для перевірки задач, може бути дуже гнучким, і можна їх реалізувати для перевірки задачі будь якої складності.

Розпочнемо з методу getOutput, який повертає генератор, і власне, потік даних на output.

```
export class ExampleValidator implements Checker {
  private readonly cases = [
    {
      input: [1, 1],
      output: 2,
    },
    {
      input: [10, 5],
      output: 15,
    },
    {
      input: [1000000000, 1],
      output: 1000000001,
    },
  ];
  *getInput(caseId: number): Generator<Promise<string> | string> {
    const cases = this.cases[caseId];
    for (const item of cases.input) {
      yield item.toString();
    }
  }
```

Рисунок 3.9 огляд валідатора

```
  isCorrect(
    caseId: number,
    output: string[],
  ): Promise<ResultChecker> | ResultChecker {
    const cases = this.cases[caseId];
    const firsLine = Number(output[0]);
    if (firsLine === cases.output) {
      return {
        isOk: true,
      };
    }

    return {
      isOk: false,
    };
  }

  getTotalCases(): Promise<number> | number {
    return this.cases.length;
  }
}
```

Рисунок 3.10 огляд валідатора

Так як ми поки реалізуємо систему, ізолювавшись від створення DAPP, додамо нашу задачу вручну. І запустимо перевірку.

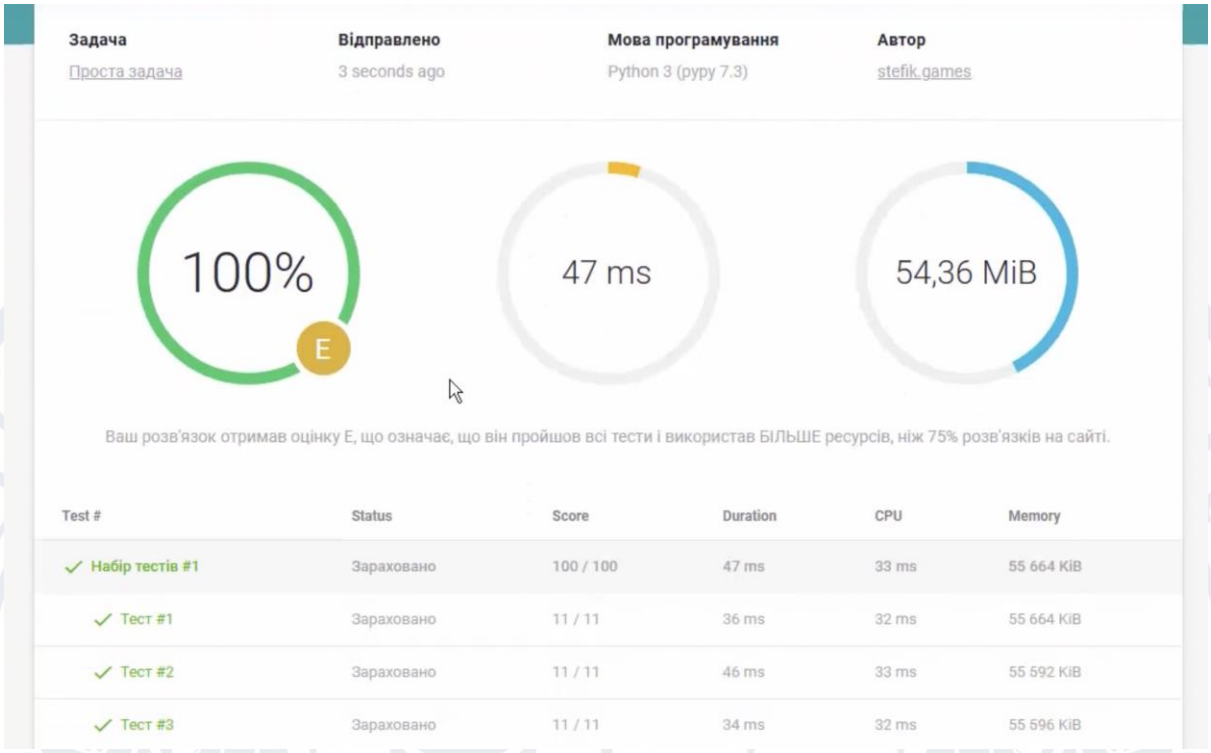


Рисунок 3.11 запуск задачі

3.3 Написання додатку, який відповідає за інтеграцію з блокчейном

Приступимо до написання частини, яка відповідає за інтеграцію з блокчейном. Ключовими моментами тут буде написання UI частини з використанням web 3.0, де власне, буде робота з смарт контрактом та авторизація користувача, та написання DAPP, де будуть відбуватись розрахунки, та запис у смарт контракт інформації про перевірку задач.

Розпочнемо з інтеграції з web 3.0 з боку front-end частини, найпопулярнішим провайдером та гаманцем для ефіріум блокчейну є Metamask, для початку реалізуємо підтримку самого його. Metamask є найобільш розповсюдженим інструментом для надсилання та отримання підписаних транзакцій. MetaMask — це криптогаманець і шлюз до блокчейну DAPP. Він використовується для підключення нашої програми до web3. Це просто розширення Chrome, яке використовується для доступу та взаємодії з блокчейном Ethereum. Його функції підтримують токени та цифрові активи в екосистемі Ethereum. Він також використовується як основний гаманець для зберігання балансу в Ethereum.

Для підключення ми використовуємо ethers.js, щоб підключитися до гаманця Ethereum. Metamask додає у глобальний об’єкт window ключ Ethereum, через який, власне, і буде відбуватись вся комунікація з блокчейном, це в якомусь роді, провайдер, який надає можливість інтегруватись у блокчейн, і реалізовувати web 3.0.

Для більш зручної комунікації використаємо бібліотеку ether.js. Розглянемо, безпосередньо, те, яким чином можна проводити маніпуляції з смарт контрактом

```
async function createTask(text: string, price: string) {
  const abi = [
    "function createTask(string memory text, uint256 price) external",
    ];
  const web3Modal = new Web3Modal({
      cacheProvider: true,
      providerOptions: {
          walletconnect: {
              package: WalletConnectProvider,
          }
        },
  });
  const instance = await web3Modal.connect();
  const provider = new ethers.providers.Web3Provider(instance);
  const signer = provider.getSigner();
  const smartContract = new ethers.Contract(contractAddress, abi, provider);
  const contractWithSigner = smartContract.connect(signer);
  const tx = await contractWithSigner.post(text, ethers.utils.parseEther(price));
  await tx.wait();
}
```

Рисунок 3.12 інтгреація з смарт контрактом

У представленому сніпеті, ми запитуємо авторизацію у користувача, тобто, можливість виконувати певні операції, які потребують інформації про користувача. При виконанні цього коду відкривається вікно

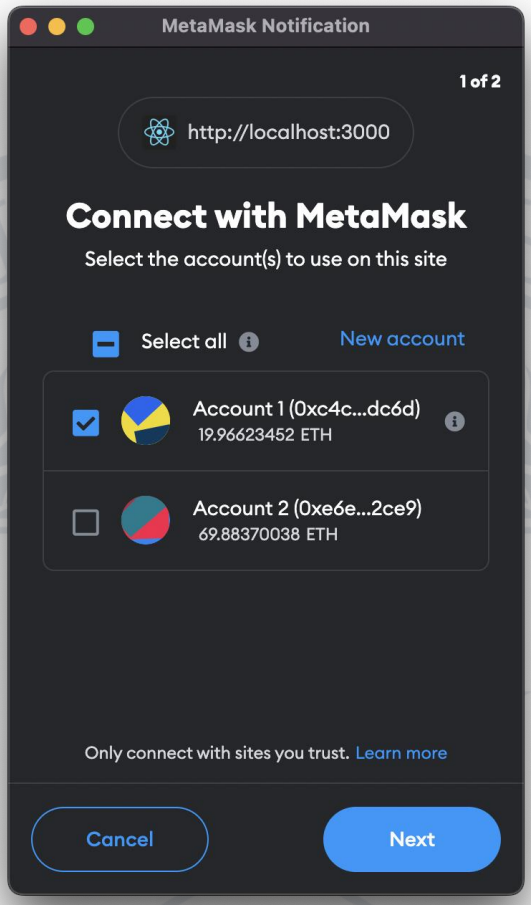


Рисунок 3.13 вікно підключення до Metamask.

За рахунок цього вікна можна надати додатку дозвіл використовувати власний гаманець, переглядати баланс рахунку, та виконувати операції з смарт контрактами чи надавати транзакції на підпис.

Отже передавши цю інформацію до нашого API, з'являється можливість записувати інформацію у блокчейн для певного користувача, і в подальшому реалзіувавши механізм Redeem оримувати винагороду.

3.4 Отримання нових задач

Для створення, так званого, дослідника блокчейну, для пошуку нових задач використаємо концепцію підграфу.

Підграф, в контекті The Graph це невеличкий проект, що містить в собі інформації про можливі запити, та потрібні дані для індексування. Для створення проекту потрібно встановити graph-cli з домогою команди npm install -g @graphprotocol/graph-cli.

І далі сгенерувати базовий проект, використавши команду

graph init

Після цього буде створено

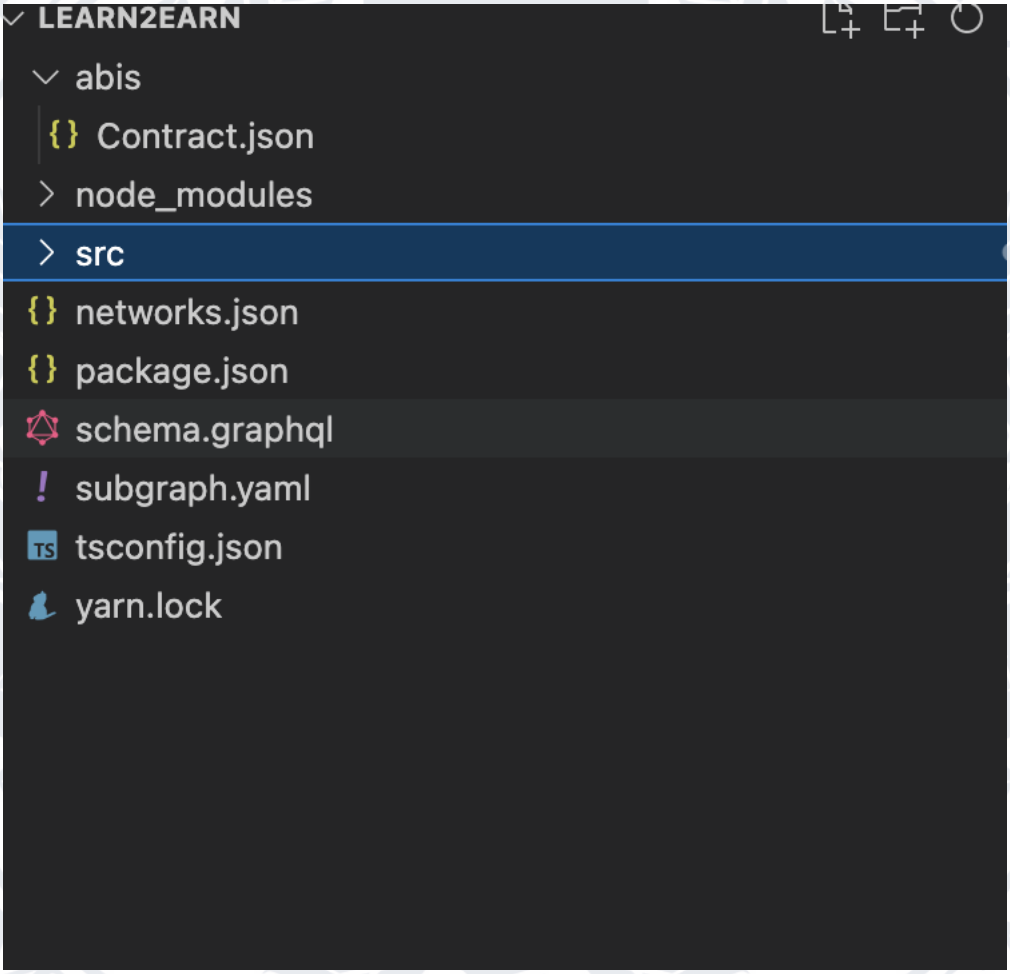


Рисунок 3.14 subgraph проект

Розберемось що за що відповідає

- Abi – папка яка містить бінерні інтерфейси смарт контракту, тобто смарт контракт компілюється у бінарний код, і для взаємодії з ним і визову команд з бінарного коду потрібно abi

- Src – папка де міститься код сабграфу, з інформацією про те що необхідно індексувати
- Networks.json – файл містить інформацію про те у якій мережі розгорнути смарт контракт та яку адресу має
- Schema.graphql – містить інформацію про те які запити повинен повнен мати сабграф.
- Subgraph.yaml – файл-маніфест від thegraph, відповідно

Для того щоб реалізувати так званий сабграф, нам необхідно

- отримати abi смарт контракту, за рахунок якого відбувається отримання інформації про події як виникають у контракт
- далі у маніфесті нам потрібно описати відповдність події до обробника
- в обробнику описати логіку за якою буде відбуватись зберігання інформації

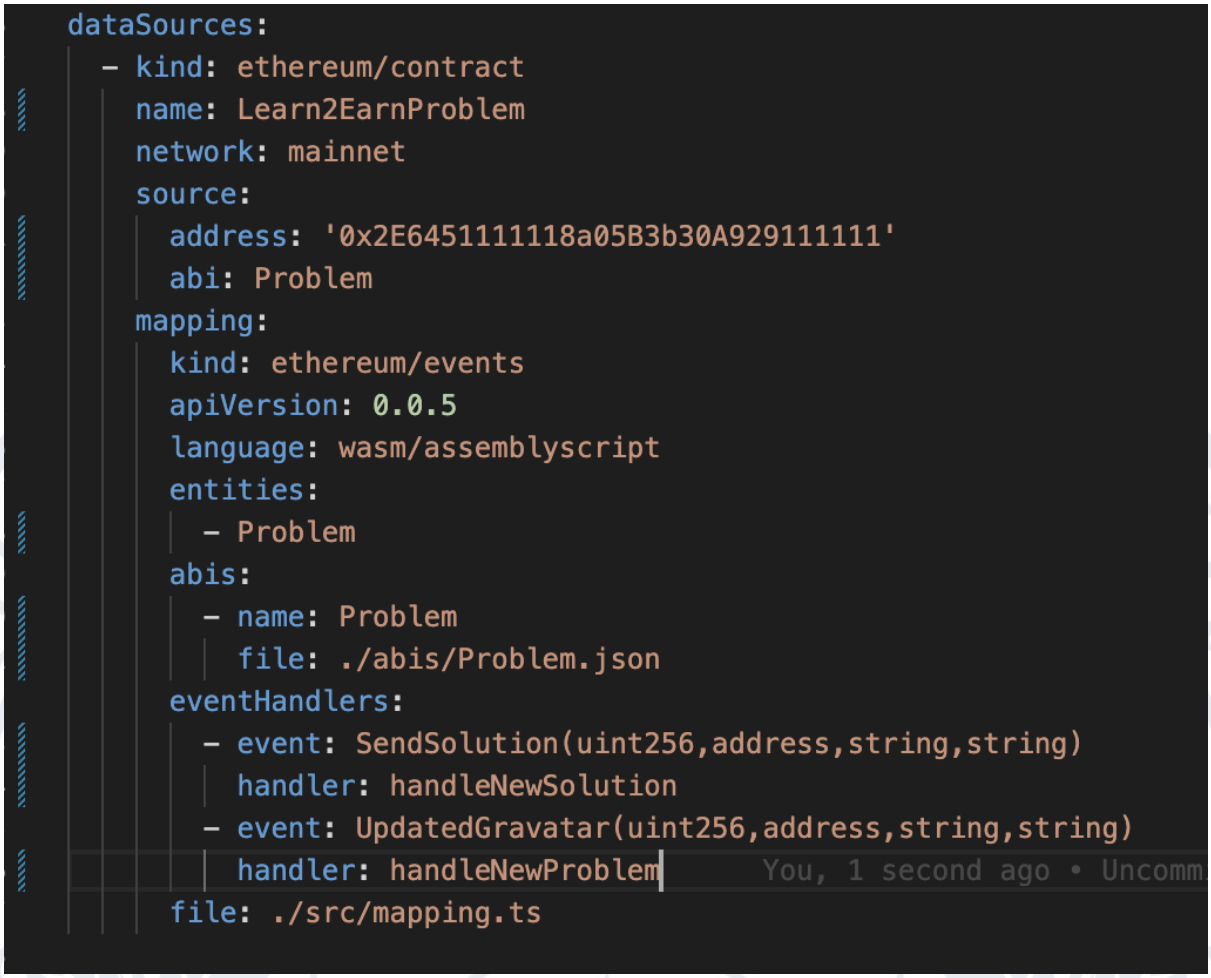


Рисунок 3.15 огляд маніфесту

Далі приступимо до написання обробників. В спрощеному вигляді це виглає наступним чином.

```
export function handleNewSolution(event: NewSolutionEvent): void {
  let gravatar = new Solution(event.params.id.toHex())
  gravatar.owner = event.params.owner
  gravatar.displayName = event.params.displayName
  gravatar.problemUrl = event.params.problemUrl
  gravatar.save()
}

export function handleNewProblem(event: NewProblemEvent): void {
  let id = event.params.id.toHex()
  let gravatar = Problem.load(id)
  if (gravatar == null) {
    gravatar = new Problem(id)
  }
  gravatar.owner = event.params.owner
  gravatar.displayName = event.params.displayName
  gravatar.problemUrl = event.params.problemUrl      You, 1 second a
  gravatar.save()
}
```

Рисунок 3.16 огляд обробників

Отже тут ми можемо отримувати інформацію про події спроби здачі розв’язання, та створення нової задачі, та зберігати це в себе.

3.5 Висновок

В ході виконання роботи з використанням технологій розробки DAPP для перевірки задач було реалізовано сервіс, котрий надає кінцевому користувачу можливість геймифікувати процес навчання спортивному програмуванню. За рахунок чого розв’язуючі задачі можна отримувати винагороду у вигляді токену.

Реалізовано наступні можливості:

- Надсилання сирцівного коду на перевірку на певному набору тестів
- Можливість легкого розширення набору задач
- Можливість в режимі реального часу слідкувати за етапами виконання перевірки
- Можливість легкого масштабування та розширення функціоналу сервісу.

ВИСНОВКИ

Дана робота розроблялася з метою популяризації технологій блокчейну та спортивного програмування, за рахунок створення сервісу, який надасть можливість легкої геймифікації процесу навчання та розв’язування задач.

У розробленому продукті було реалізовано наступні можливості:

- Надсилання сирцівного коду на перевірку на певному набору тестів
- Можливість легкого розширення набору задач
- Можливість в режимі реального часу слідкувати за етапами виконання перевірки
- Можливість легкого масштабування та розширення функціоналу сервісу.

У першому розділі було проведено аналіз схожих продуктів, з точки зору підходу до реалізації системи перевірки задач та використання підходу web 3.0. У другому розділі ми ознайомились з технологіями котрі будемо використовувати при розробці відповідного додатку. Відповідно, у третьому розділі, було продемонстровано безпосередньо розробку додатку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Web 3.0?: [Електронний ресурс] – Режим доступу до ресурсу: https://coinmarketcap.com/alexandria/article/what-is-web-3-0
2. Nest JS Documentation?: [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nestjs.com/
3. RabbitMQ [Електронний ресурс] – Режим доступу до ресурсу: https://www.rabbitmq.com/documentation.html
4. TypeORM [Електронний ресурс] – Режим доступу до ресурсу: https://typeorm.io/
5. A Quick Guide To Understanding RabbitMQ & AMQP [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/swlh/a-quick-guide-to-understanding-rabbitmq-amqp-ba25fdfe421d
6. A complete guide to PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://prabhupant.medium.com/a-complete-guide-to-postgresql-e4d1cefb9866
7. My TypeScript Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://non-traditional.dev/my-typescript-best-practices-845e196284aa?gi=5efc963d1fe9
8. The Graph – офіційний ресурс [Електронний ресурс] - Режим доступу до ресурсу: https://thegraph.com/en/
9. Ethers.JS – документація [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ethers.io/v5/
10. Top 10 Web3 Applications You Must Know – офіційний ресурс [Електронний ресурс] - Режим доступу до ресурсу: https://101blockchains.com/top-web3-applications/
11. How to… avoid common mistakes in dApp development – офіційний ресурс [Електронний ресурс] - Режим доступу до ресурсу:

https://medium.com/wavesprotocol/how-to-avoid-common-mistakes-in-dapp-development-61015e700459

12. NestJS Docs [Електронний ресурс] - Режим доступу до ресурсу: https://docs.nestjs.com/first-steps
13. INTRODUCTION TO DAPPS [Електронний ресурс] - Режим доступу до ресурсу: https://ethereum.org/en/developers/docs/dapps/
14. Top 5 decentralized app development frameworks [Електронний ресурс] - Режим доступу до ресурсу: https://blog.logrocket.com/top-5-decentralized-app-development-frameworks/
15. Remix Docs [Електронний ресурс] - Режим доступу до ресурсу: https://ethereum.github.io/remix-ide/

Рогожук Назар Вячеславович
Прізвище, ім’я, по-батькові

Факультет інформаційних і прикладних технологій
Факультет

122 Комп’ютерні науки
Шифр і назва спеціальності

Комп’ютерні технології обробки даних
Освітня програма

ДЕКЛАРАЦІЯ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (бакалаврська) робота на тему: «Розробка і дослідження децентралізованого медіа додатку на базі технології Blockchin» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних вправ, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувались іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна (бакалаврська) робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

дата підпис здобувача