

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу
Кафедра системного проектування**

До захисту допущено:

В. о. завідувача кафедри

_____ Анатолій Петренко

«__» _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інтелектуальні сервіс-орієнтовані
розподілені обчислювання»**

спеціальності 122 «Комп'ютерні науки»

**на тему: «Інструментарій для використання технології блокчейн у веб-
розробці»**

Виконав (-ла):

студент (-ка) IV курсу, групи ДА-71

Павлюк Вадим Дмитрович _____

Керівник:

к.т.н. доцент кафедри, Булах Богдан Вікторович _____

Консультант з економічного роділу:

к.е.н, доцент Рощина Надія Василівна _____

Консультант з нормоконтролю:

доцент, Кирюша Богдан Анаталійович _____

Рецензент:

в.о. зав. каф. ММСА, к.т.н., доц. Тимошук О. Л. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Анатолій Петренко

«___» _____ 20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Павлюку Вадиму Дмитровичу

1. Тема роботи «Інструментарій для використання технології блокчейн у веб-розробці», керівник роботи Булах Богдан Вікторович, к.т.н. Доцент кафедри, затверджені наказом по університету від «___» _____ 20__ р. № _____
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи
4. Зміст роботи
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	к.е.н, доцент Рощина Надія Василівна		

7. Дата видачі завдання 27 березня 2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вивчення літератури за темою	12.04.2021	
2	Підготовка першого розділу	19.04.2021	
3	Підготовка другого розділу	30.04.2021	
4	Підготовка третього розділу	10.05.2021	
5	Реалізація централізованого веб-додатку	15.05.2021	
6	Реалізація децентралізованого веб-додатку	20.05.2021	
7	Підготовка четвертого розділу	25.05.2021	
8	Підготовка економічної частини	28.05.2021	
9	Оформлення розділів відповідно до нормоконтролю	31.05.2021	
10	Підготовка презентації	05.06.2021	

Студент

Вадим Павлюк

Керівник

Богдан Булах

РЕФЕРАТ

Дипломна робота: 90 ст., 76 рис, 6 табл., 2 дод., 16 джерел.

БЛОКЧЕЙН. ЦЕНТРАЛІЗОВАНИЙ ДОДАТОК. ДЕЦЕНТРАЛІЗОВАНИЙ ДОДАТОК.

Тема: Інструментарій для використання технології блокчейн у веб-розробці.

Об'єкт дослідження: блокчейн.

Предмет дослідження: використання технології блокчейн у веб-розробці.

Мета роботи: дослідити блокчейн та проаналізувати інструменти блокчейну для веб розробки. Порівняти централізовані та децентралізовані системи та реалізувати веб-додаток.

Створено веб-додаток Todo List двома способами:

- Централізований(стек технологій: Vue, Firebase)
- Децентралізований(стек технологій: Vue, Metamask, Ganache, Solidity, Truffle)

ABSTRACT

Thesis: 90 pages, 76 figures, 6 tables, 2 appendices, 16 sources.

BLOCKCHAIN. CENTRALIZED APP. DECENTRALIZED APP.

Topic: Tools for using blockchain technology in web development.

Object of research: blockchain.

Subject of research: use of blockchain technology in web development.

Purpose: explore blockchain and analyze blockchain tools for web development. Compare centralized and decentralized systems and implement a web application

The Todo List web application was created in two ways:

- Centralized (technology stack: Vue, Firebase)
- Decentralized (technology stack: Vue, Metamask, Ganache, Solidity, Truffle)

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1 ОСНОВНІ СКЛАДОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН	10
1.1 Що таке блокчейн.....	10
1.2 Структура блоку	10
1.3 Ланцюг блоків	12
1.4 P2P мережа.....	14
1.4 Процес транзакції блокчейн	15
1.5 Приклад транзакції.....	16
1.5.1 Фізична транзакція	16
1.5.2 Цифрова транзакція	16
1.5.3 Книга цифрових транзакцій.....	17
1.5.4 Децентралізована книга.....	18
1.6 Переваги та недоліки	19
1.6.1 Переваги	19
1.6.2 Недоліки.....	21
1.7 Версії блокчейн.....	22
1.8 Історія Блокчейну	23
1.8.1 Ідея	23
1.8.2 Багаторазовий доказ роботи(RPoW).....	23
1.8.3 Біткоїн-мережа.....	24
1.8.3 Ефіріум.....	24
1.9 Висновки до розділу	25
РОЗДІЛ 2 ЦЕНТРАЛІЗОВАНІ ТА ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ.....	26
2.1 Централізована система	26
2.2 Децентралізована система.....	27
2.3 Приклади.....	28
2.3.1 Платіжна система	28
2.3.2 Вибори президента.....	29
2.3.3 Електроенергія.....	30

2.4 Висновки до розділу	32
РОЗДІЛ 3 ІНСТРУМЕНТИ БЛОКЧЕЙНУ ДЛЯ ВЕБ-РОЗРОБКИ	33
3.1 Додатки з використанням блокчейна.....	33
3.1.1 Використання платформи для блокчейн-додатків.....	33
3.1.2 Власна мережа на базі open-source блокчейн-платформи	35
3.1.3 Створення проекту з використанням блокчейна з нуля	36
3.2. Інструменти блокчейну для веб розробки	37
3.2.1 Geth	38
3.2.2 Remix IDE	38
3.2.3 Mist	39
3.2.4 Truffle	39
3.2.5 Metamask	40
3.2.6 Parity	40
3.2.7 Ganache.....	41
3.2.8 Embark.....	41
3.3 Висновки до розділу	42
РОЗДІЛ 4 ПРОГРАМНА РЕАЛІЗАЦІЯ.....	43
4.1 Веб-додаток Todo-List	43
4.2 Технології для централізованого додатку.....	43
4.2.1 Vue	43
4.2.2 Firebase	44
4.3 Технології для децентралізованого додатку.....	44
4.3.1 Truffle	44
4.3.2 Metamask	45
4.3.3 Ganache.....	46
4.3.4 Solidity	46
4.4 Архітектура децентралізованого додатку	47
4.4.1 Web3js та Web3 Provider	48
4.4.2 Смарт-контракт.....	48
4.5 Архітектура централізованого додатку	49

4.6 Налаштування середовища для децентралізованого	49
4.6.1 Установка Truffle	49
4.6.2 Установка Metamask	55
4.6.3 Установка Vue	58
4.7 Налаштування середовища для централізованого	59
4.7.1 Установка Firebase	59
4.8 Результат роботи додатку	62
4.8.1 Створення нового завдання	63
4.9 Висновки до розділу	65
5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	66
5.1 Постановка задачі проектування	66
5.2 Обґрунтування функцій та параметрів програмного продукту	66
5.3 Обґрунтування системи параметрів програмного продукту	68
5.4 Аналіз експертного оцінювання параметрів	70
5.5 Економічний аналіз розробки програмного забезпечення	73
5.6 Вибір кращого варіанту	76
Висновки до розділу	76
ВИСНОВКИ ПО РОБОТІ	77
ПЕРЕЛІК ПОСИЛАНЬ	78
ДОДАТОК А	80
ДОДАТОК Б	86

ВСТУП

Інформаційні технології з'являються і розвиваються настільки стрімко, що фахівці часом не встигають зреагувати на їх появу, що і сталося з блокчейн. Тому без знання специфіки даної технології, буде досить проблематично орієнтуватися в економіці та сфері фінансів, а також в програмуванні в частості розробки веб-додатків. Крім того, активно обговорюється використання технології - блокчейн в якості бази для платіжних систем, де центральні банки в подальшому можуть її використовувати для національної валюти.

Технологія блокчейн надійна, оскільки зберігає однакові блоки інформації у своїй мережі. З цієї причини блокчейн не може бути під контролем будь-якого окремого суб'єкта, і він не має жодної точки відмови. Зберігаючи дані в мережі, він виключає ризики, пов'язані з централізованим зберіганням даних. У мережах відсутні централізовані точки вразливості, які комп'ютерні хакери можуть легко використати.

Першим додатком на базі технології блокчейн став біткоїн для однойменної криптовалюти. В останні роки він послужив основою для створення інших блокчейн-додатків. Біткойн-блокчейн - це один з найбільших і найпотужніших блокчейнів у світі. Був призначений в основному для відправки біткойна, криптовалюти.

Оскільки блокчейн є настільки затребуваною та універсальною технологією, низка великих технологічних фірм прийняли його і почали пропонувати блокчейн як послугу. До них належать:

- Amazon Web Services (один із наших найкращих хостингових партнерів)
- Служба блокчейнів Microsoft Azure
- Платформа IBM Blockchain

Зростання цих послуг означає, що блокчейн стає все більш широко використовуваним.

РОЗДІЛ 1 ОСНОВНІ СКЛАДОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

1.1 Що таке блокчейн

Blockchain - це децентралізована цифрова книга (постійно зростаючий список електронних записів) транзакцій, що зберігаються і захищаються за допомогою криптографії. Дані книги блокчейну розподіляються по мережі комп'ютерів. Користувачі можуть безпосередньо взаємодіяти із збереженими даними в режимі реального часу, не потребуючи посередника для автентифікації транзакцій. Технологія забезпечує незалежну, стійку до втручання та прозору платформу для сторін у блокчейні для надійного зберігання, передачі та обробки конфіденційної інформації. Оскільки, після того, як блок даних записано в цифрову книгу, змінити чи видалити надзвичайно складно. Коли потрібно додати до нього якісь дані то, учасники мережі маючи копії існуючого блокчейну, запускають алгоритми для оцінки та перевірки запропонованої транзакції. Якщо більшість вузлів погодиться, що транзакція виглядає дійсною - тобто ідентифікаційна інформація відповідає історії блокчейну - тоді нова транзакція буде схвалена і новий блок додається до ланцюжка[1].

1.2 Структура блоку

Блокчейн - це послідовний ланцюг блоків які зберігають дані. Кожен блок(рис. 1.1) має такі поля:

- хеш блоку(рис. 1.2)
- дані(рис. 1.3)
- хеш попереднього блоку дані(рис. 1.4)

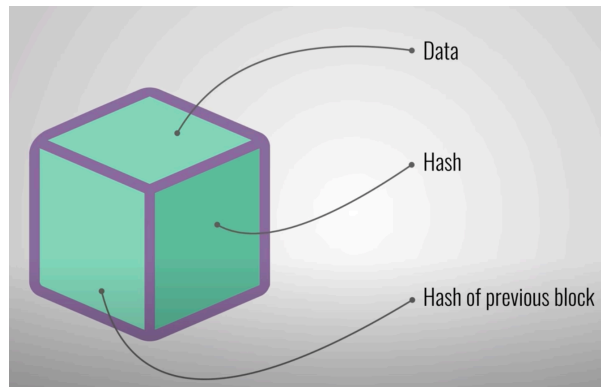


Рисунок 1.1 - кожен блок містить дані, хеш і хеш попереднього блоку

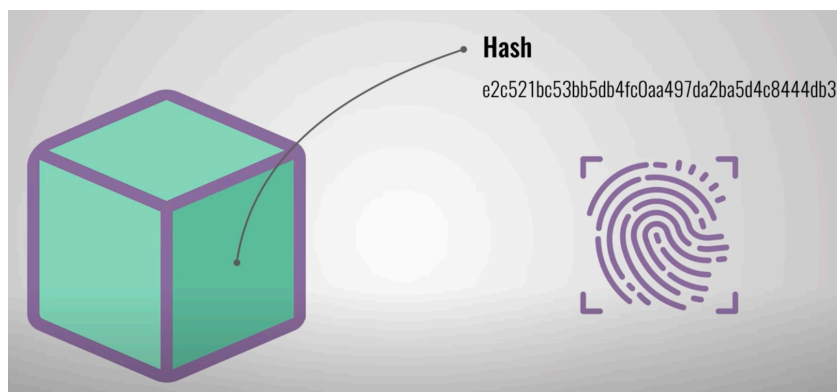


Рисунок 1.2 - хеш ідентифікує блок і завжди унікальний, як тільки блок створений, його хеш розраховується. При спробі, змінити щось усередині блоку, призведе до зміни хешу.

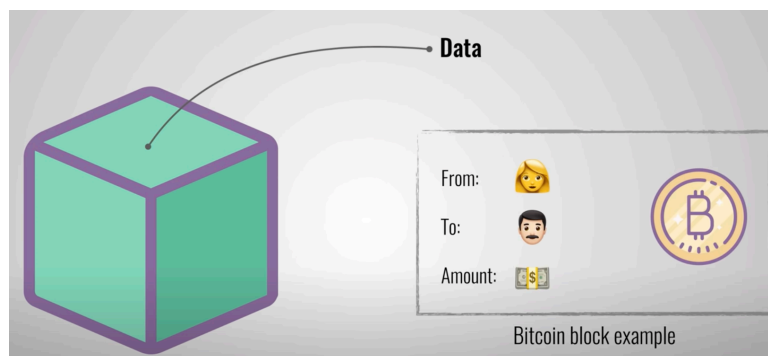


Рисунок 1.3 - дані, що зберігаються всередині блоку, залежать від типу блоку. Наприклад Bitcoin Blockchain, зберігає деталі про транзакції. (Відправник, одержувач і кількість монет)

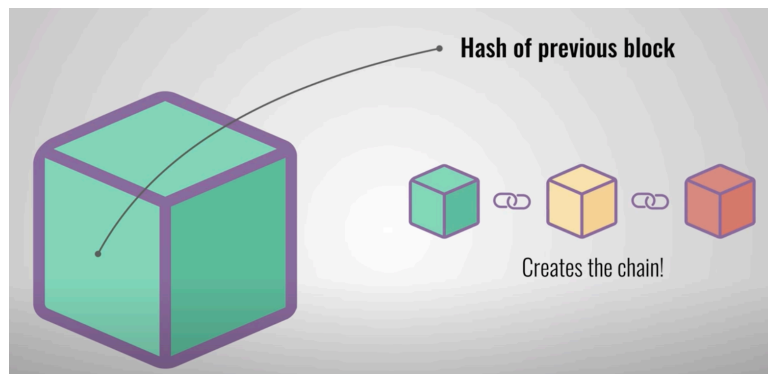


Рисунок 1.4 - хеш попереднього блоку, ефективно створює ланцюжок блоків.

1.3 Ланцюг блоків

Розглянемо простий приклад, як зв'язані блоки між собою(рис. 1.5)

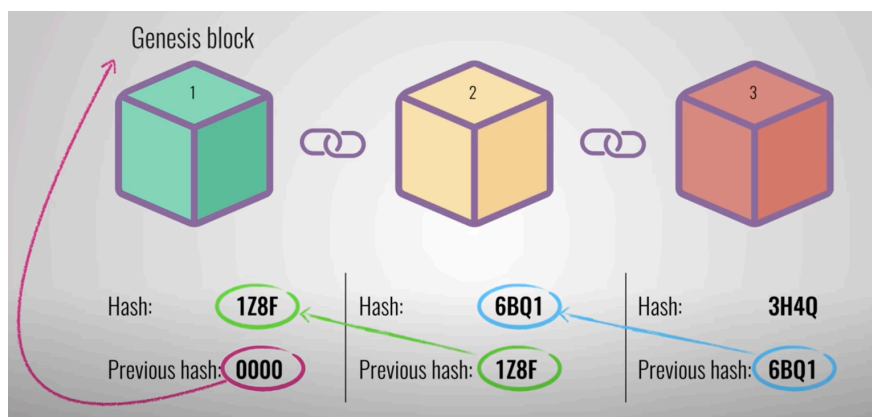


Рисунок 1.5 - приклад дійсного ланцюга блокчейна

З малюнка видно, що перший блок називається “Генезіс блок”(Genesis block) пов'язано це з тим, що хеш попереднього блоку дорівнює 0. Блоки формують ланцюг за допомогою хеша попереднього блоку.

Попередній хеш третього блоку вказує на хеш другого блоку і в свою чергу попередній хеш другого блоку вказує на хеш першого блоку. Ланцюг називається дісним, оскільки кожний блок зберігає правильний хеш попереднього блоку крім першого.

Що буде якщо змінити дані в другому блоці(рис. 1.6)?

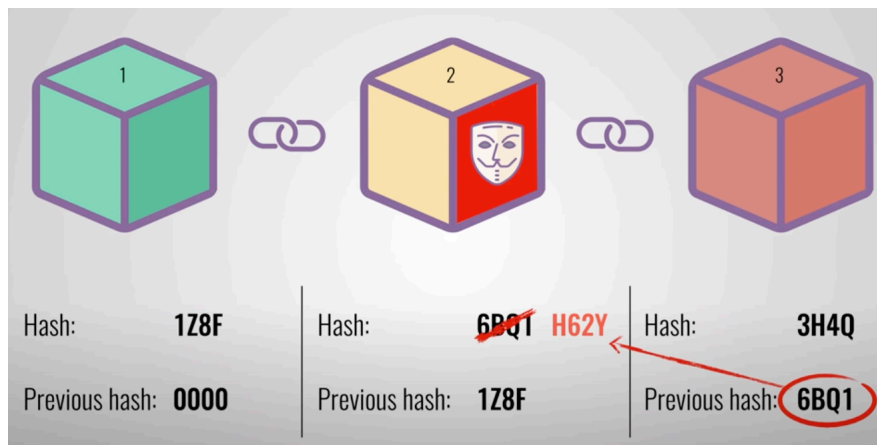


Рисунок 1.6 - приклад не дійсного ланцюга блокчейна

Як видно з малюнку хеш другого блоку став іншим це пов'язано з тим, що дані які зберігає блок змінилися, тому хеш перерахувався. В третьому блоці зберігається неправильний попередній хеш, тому ланцюг не дійсний. Як зробити ланцюг знову дійсним?

Існує механізм Proof of work(доказ роботи) - який уповільнює створення нових блоків та перерахунок хеша(якщо змінили дані у блоці). У нашому випадку перерахуємо хеш другого та третього блоків та всіх наступних(оскільки після перерахунку хеш у третьому блоці зміниться, отже попередній хеш у четвертому блоці буде не правильним, тому його перерахують і так далі). Чому механізм уповільнює створення та перерахунок?

Візьмемо BitCoin блокчейн, створення нового або перерахунок хеша блоку триває 10 хвилин(рис. 1.7).

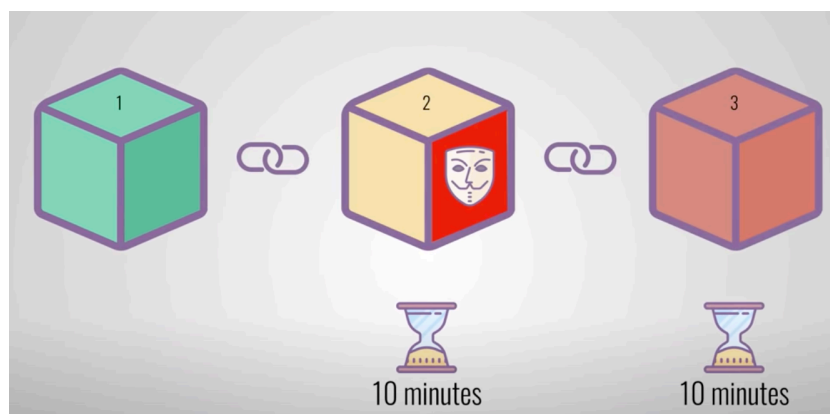


Рисунок 1.7 - перерахунок хешу

Це пов'язано з тим, що є параметр “складність”, нехай він дорівнює п'яти. При створенні нового блоку розраховується хеш, так як “складність п'ять”, то хеш повинен починатися з п'яти нулів. Поки не виконається умова, хеш буде перераховуватись із-за цього час створення блоку зростає. У нашому випадку це буде тривати 20 хвилин(перерахунок другого та третього блоків).

1.4 P2P мережа

Peer-to-Peer(P2P) мережа - це комп'ютери, які беруть участь у одноранговій мережі тобто всі вони рівні, немає головних вузлів(серверів). Вузли в мережі P2P як надають, так і споживають послуги одночасно із взаємністю, яка діє як стимул до участі. Мережі P2P за своєю суттю є стійкими, децентралізованими та відчиненими[2]. Розглянемо приклад такої мережі(рис. 1.8)

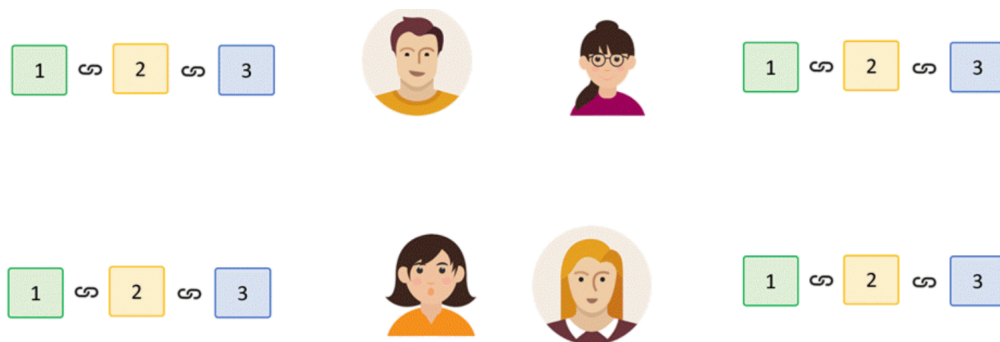


Рисунок 1.8 - p2p мережа[3]

Коли хтось увійде в цю мережу, то отримає повну копію блокчейну. Кожен комп'ютер називається вузлом. Подивимось, що відбувається, коли будь-який користувач створює новий блок(рис 1.9). Цей новий блок надсилається всім користувачам мережі. Кожен вузол повинен перевірити блок, щоб переконатися, що він не був змінений. Після повної перевірки кожен вузол додає цей блок до свого блокчейну[3].

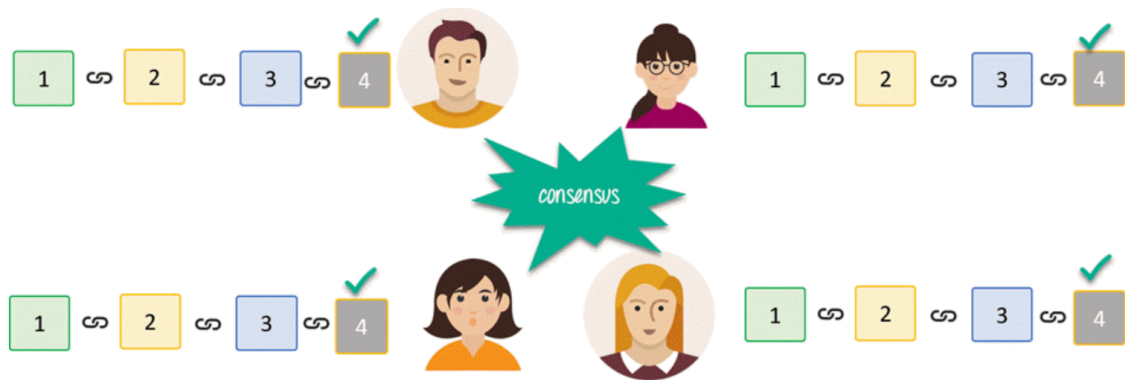


Рисунок 1.9 - додавання нового блоку в р2р мережу[3]

Усі вузли в цій мережі створюють консенсус. Вони домовляються про те, які блоки є дійсними, а які ні. Вузли в мережі будуть відхиляти блоки, які були модифіковані. Для успішного втручання в blockchain:

- потрібно втрутитися в усі блоки на ланцюзі
- виконати proof-of-work механізм для всіх блоків
- взяти під свій контроль більше 50% однорангової мережі

Після всього цього фальсифікований блок стане прийнятим усіма іншими. Це майже неможливе завдання, тому блокчейн настільки безпечний[3].

1.4 Процес транзакції блокчейн



Рисунок 1.10 - процес транзакції[3]

Кроки(рис. 1.10):

- 1) Деяка особа вимагає транзакції. У транзакції може бути задіяна криптовалюта, контракти, записи чи інша інформація[3].
- 2) Запитувана транзакція транслюється до мережі P2P за допомогою вузлів[3].

- 3) Мережа вузлів перевіряє транзакцію та статус користувача за допомогою відомих алгоритмів[3].
- 4) Після завершення транзакції новий блок додається до існуючого блокчейну. Таким чином, що є постійним і незмінним[3].

1.5 Приклад транзакції

1.5.1 Фізична транзакція

Є Аліса та Боб, Аліса передає фізичну фішку казино Бобу. Боб тепер має її, а Аліса ні, транзакція(рис. 1.11) завершена. Їм не потрібно звертатися до посередника для перевірки транзакції. Аліса уже не може дати комусь фішку казино, оскільки дала Бобу[4].



Рисунок 1.11 - фізична транзакція[4]

1.5.2 Цифрова транзакція

Але, якщо ця транзакція(рис. 1.12) відбувається онлайн та фішка казино уже не фізична, а цифрова(ряд одиниць та нулів). Аліса передасть її Бобу, наприклад, по електронній пошті. Боб отримає цифрову фішку, а Аліса втратить. Нажаль так це не спрацює, оскільки Аліса могла зробити безліч копій цифрової фішки та відправити кому завгодно або розмістити в інтернеті, щоб усі скачували[4].

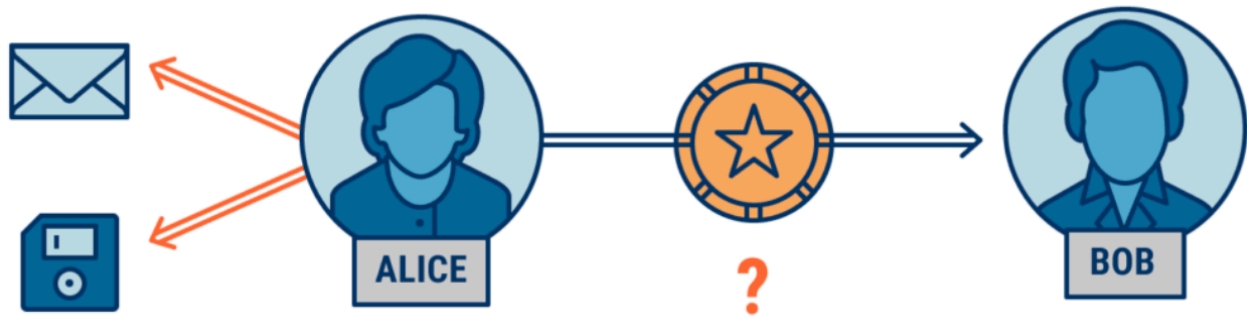


Рисунок 1.12 - цифрова транзакція[4]

Хто після цього являється власником цифрової фішки не зрозуміло.

1.5.3 Книга цифрових транзакцій

Книга цифрових транзакцій у ній будуть зберігатися транзакції. Коли Аліса передає цифрову фішку Бобу то книга фіксує цю транзакцію. Але є проблема, хто буде вести цю книгу. Вони не можуть контролювати книгу, оскільки кожен із них може збрехати, наприклад, Аліса може стерти транзакцію і сказати, що вона все ще володіє фішкою, хоча вона передала її Бобу. Це також не може бути Боб, оскільки він міг змінити транзакцію і брехати, кажучи, що Аліса дала йому дві фішки[4].

У цій ситуації потрібен посередник, наприклад, Дейв. Аліса та Боб будуть довіряти йому, а він в свою чергу буде контролювати та оновлювати цифрову книгу(рис. 1.13).

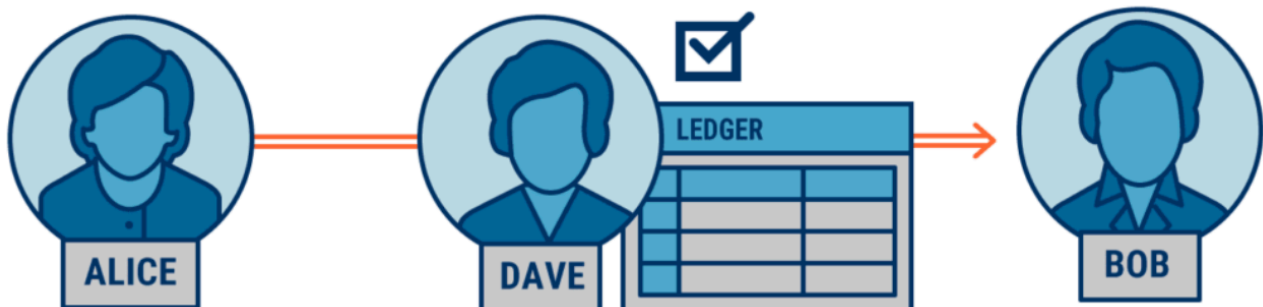


Рисунок 1.13 - книга цифрових транзакцій

1.5.4 Децентралізована книга

Книга цифрових транзакцій вирішила проблему. Аліса не зможе відправити Бобу та одночасно зберегти собі одну цифрову фішку, але, якщо Дейв захоче стягувати комісію за транзакцію або Аліса може підкупити його, щоб він стер транзакцію або вони не довіряють Дейву як посереднику. У даному випадку рішення наступне. Аліса та Боб могли роздати книгу всім своїм довіреним друзям, а не лише Дейву, і децентралізувати довіру(рис. 1.14). Оскільки книга є цифровою, усі копії книги можуть синхронізуватися разом. Якщо більшість учасників погодиться, що транзакція є дійсною (наприклад, підтвердити, що Аліса насправді володіє цифровою фішкою казино, яку вона хоче надіслати), то вона додається до книги[4].



Рисунок 1.14 - децентралізована книга[4]

Коли у багатьох людей є копія тієї ж книги, обдурити стає значно складніше. Якщо Аліса або Боб хотіли модифікувати транзакцію, їм довелося б скомпрометувати більшість учасників, що набагато складніше, ніж компрометувати одного учасника[4].

Аліса не може стверджувати, що ніколи не надсилала цифровий фішку Бобу, оскільки її копія книги не погодилась з усіма іншими. Боб не міг стверджувати, що Аліса дала йому дві цифрові фішки, оскільки його копія книги не синхронізувалася. І навіть якщо Аліса підкупує Дейва, щоб він змінив свою копію книги, Дейв тримає лише одну копію книги, а думка більшості показує, що цифровий фішка була відправлена[4].

1.6 Переваги та недоліки

1.6.1 Переваги

1. Ефективна та швидка обробка транзакцій. Блокчейн забезпечує швидку обробку транзакцій, в той час як за звичайною схемою цей процес може зайняти кілька днів[5].
2. Безпека та надійність. Протягом багатьох років система зазнавала численних атак та різноманітних загроз, але жодна з них не мала успіху. Отже, блокчейн - це практично захищена від злому система, стійка до спроб злому, крадіжки чи підробки. Записи, що зберігаються в ньому, вважаються незмінними та захищеними від втручання. Це можливо завдяки:
 - a) Децентралізація. Дані транзакцій не зберігаються на головному сервері, а розподіляються по ланцюгу комп'ютерів. Зберігаючи дані в мережі, Blockchain усуває ризики, пов'язані з централізованим зберіганням даних. Оскільки в ньому відсутні точки вразливості та збою, які можуть бути використані хакерами. Отже, це забезпечує кращий опір зловмисникам та зовнішнім атакам у системі[5].
 - b) Захист даних від модифікації та видалення, що є однією з головних переваг Blockchain. Мережа фіксує всю історію транзакцій у цифровій книзі. Старі транзакції зберігаються в книзі назавжди, а нові транзакції додаються до неї в кінець. Будь-яка зміна будь-якого біта даних призведе до перерахунку значення хешу, а новий хеш не буде відповідати старому.

Отже, будь-яка спроба втручання в будь-яку частину блокчейну буде відразу очевидною[5].

- c) Система розподіленої перевірки транзакцій, яка гарантує, що ніхто не може втручатися у записи, а кожен у мережі може перевірити книгу та побачити таку ж історію транзакцій, як і всі інші[5].
- d) Використання відкритих та приватних ключів. Відкритий ключ (довгий, випадково сформований рядок чисел) - це адреса користувача в блокчейні, тоді як приватний ключ - це пароль, який надає користувачеві доступ до його цифрових активів і дозволяє взаємодіяти з різними можливостями, які підтримує Blockchain[5].

- 3. Не потрібно залучати посередників (треті сторони, яким довіряють), таких як повільні та дорогі платіжні системи, брокери, агенти, довідкова служба або допомога бек-офісу, для обробки транзакцій. Це можливо, оскільки Blockchain є загальнодоступним і добре розробленим, щоб замінити функції, які в даний час покладені на посередників, такі як обробка фінансових операцій, забезпечення надійного торгового середовища, забезпечення дотримання контрактів або захист від шахрайства. Як результат, транзакції можуть бути швидшими, дешевшими, безпечнішими та простішими у доступі[5].
- 4. Скорочення витрат. Уникаючи використання посередників та накладних витрат для обміну активами, блокчейн може потенційно заощадити гроші[5].
- 5. Прозорість та відкритість. Інформацію, що міститься в блоках, можна швидко перевірити, оскільки кожен може отримати доступ до книги та переглянути перевірені транзакції[5].

6. Точність. Дані не потрапляють до системи Blockchain, якщо вони не перевірені. Отже, дані в Blockchain є точними, повними, послідовними та своєчасними[5].
7. Широко застосовна технологія. Оскільки Blockchain задовольняє потребу в надійних записах і має великий потенціал для розвитку, його можна використовувати в різних галузях промисловості, що виходять за рамки криптовалют. Спроби використання блокчейн включають фінансовий сектор, банківську діяльність, страховий бізнес, державне управління та управління бізнесом, кампанії масового фінансування, виконання та виконання контрактів, операції з нерухомістю, роздрібну торгівлю та виробництво, охорону здоров'я, продаж музики та навіть організацію демократичних виборів[5].

1.6.2 Недоліки

1. Блокчейн не може зберігати великі обсяги інформації. Однак технологія розвивається, і в майбутньому цей недолік може зникнути[5].
2. Деякі люди кажуть, що теоретично це можна зламати. Однак це було б можливим лише в тому випадку, якщо 51% користувачів в ланцюжку погодилися і створили схему шахрайства та одночасно змінили дані транзакцій. З огляду на сучасний технологічний розвиток це практично неможливо[5].
3. Енергоємне використання. Завдяки своїй розподіленій природі, Blockchain вимагає постійної обчислювальної потужності в багатьох місцях, що призводить до постійного великого споживання електроенергії у всьому світі[5].

4. Універсальному впровадженню блокчейну перешкоджає його ще не встановлений статус регулювання. Використання цієї нової технології, безсумнівно, призведе до порушення централізованих установ та бюрократій, таких як банки, державні органи чи інші сторони, які здійснюють операції, і останні, безперечно, борються з нею[5].

1.7 Версії блокчейн

Блокчейн 1.0: Валюта

Впровадження DLT (технологія розподіленої книги) призвело до її першого і очевидного застосування: криптовалют. Це дозволяє здійснювати фінансові операції на основі технології блокчейн. Біткойн - найбільш яскравий приклад у цьому сегменті[3].

Блокчейн 2.0: смарт-контракти

Нові ключові концепції - це смарт-контракти, невеликі комп'ютерні програми, які «живуть» у блокчейні. Це безкоштовні комп'ютерні програми, які виконуються автоматично та перевіряють умови, визначені раніше, такі як сприяння, перевірка чи забезпечення. Він використовується як заміна традиційних контрактів[3].

Блокчейн 3.0: DApps

DApps - це аббревіатура децентралізованої програми. Він має свій серверний код, що працює в децентралізованій одноранговій мережі. DApp може мати прикладний код інтерфейсу Blockchain та користувальницькі інтерфейси, написані будь-якою мовою, яка може здійснювати виклик до його серверної бази, як традиційні програми[3].

1.8 Історія Блокчейну

Хоча блокчейн - це нова технологія, вона вже може похвалитися багатою та цікавою історією.

1.8.1 Ідея

Технології блокчейн була описана в 1991 році, коли вчені-дослідники Стюарт Хабер та В. Скотт Сторнетта представили рішення для відмітки часу у цифрових документах, щоб їх не можна було фальсифікувати[6].

Система використовувала ланцюг блоків для зберігання відмічених часом документів, який був криптографічно захищений. У 1992 році дерева Меркла були включені до проекту, тим самим система стала більш ефективна, дозволивши зібрати кілька документів в один блок. Однак ця технологія залишилася невикористаною, а патент втратив чинність у 2004 році, за чотири роки до створення біткойнів[6].

1.8.2 Багаторазовий доказ роботи(RPoW)

У 2004 році інформатик і криптографічний активіст Хел Фінні (Гарольд Томас Фінні II) представив систему під назвою RPoW, багаторазовий доказ роботи. Система працювала, отримуючи незмінний на основі Hashcash маркер підтвердження роботи, і створювала маркер, підписаний RSA, який пізніше можна передавати від людини до людини[6].

RPoW вирішив проблему подвійних витрат, зберігаючи право власності на маркери, зареєстровані на надійному сервері, який був розроблений, щоб дозволити користувачам у всьому світі перевіряти його правильність та цілісність у режимі реального часу[6].

RPoW можна розглядати як ранній прототип і важливий ранній крок в історії криптовалют.

1.8.3 Біткоїн-мережа

Наприкінці 2008 року Сатоші Накамото, публікував статтю: “Біткоїн: однорангова електронна система платежів”.

Біткоїн заснований на алгоритмі доказу роботи Hashcash, але замість використання RPoW, захист від подвійних витрат у біткойнах забезпечувався децентралізованим одноранговим протоколом для відстеження та перевірки транзакцій. Біткойни добуваються для винагороди за допомогою механізму перевірки роботи(PoW) окремими майнерами, а потім перевіряються децентралізованими вузлами в мережі[6].

3 січня 2009 року з'явився біткоїн. Перший блок біткойнів видобув Сатоші Накамото, винагорода якого становила 50 біткойнів. Перша у світі біткоїн-транзакція відбулася 12 січня 2009 року, Сатоші Накамото передав Хел Фінні 10 біткойнів[6].

1.8.3 Ефіріум

У 2013 році Віталік Бутерін, програміст та співзасновник журналу Bitcoin, заявив, що біткойну потрібна мова сценаріїв для створення децентралізованих додатків. Не вдавшись домовитись у спільноті, він розпочав розробку нової обчислювальної платформи на базі блокчейну яка включала функцію сценаріїв, які називаються смарт-контрактами[6].

Смарт-контракти - це програми або сценарії, які розгортаються та виконуються на блокчейні Ethereum, їх можна використовувати для здійснення транзакції. Смарт-контракти пишуться на мовах програмування та компілюються в байт-код, який віртуальна машина Ethereum(EVM), може читати та виконувати.

Розробники можуть створювати та публікувати додатки, що працюють всередині блокчейну Ethereum. Ці програми називаються DApps (децентралізованими додатками), і в блокчейні Ethereum вже запущено сотні

DApps, включаючи платформи соціальних медіа, азартні програми та фінансові біржі[6].

Криптовалюта Ethereum називається Ether, вона може передаватися між рахунками і використовується для сплати зборів за обчислювальну потужність, що використовується при виконанні смарт-контрактів[6].

1.9 Висновки до розділу

Блокчейн - це зв'язаний ланцюг із блоків. Блок це структура яка містить хеш, хеш попереднього блоку, список транзакцій. Було наведено дійсний та не дійсний ланцюги у блокчейні. Було показано на прикладах механізм “Proof-of-work”, “P2P мережу” та процес транзакції блокчейну. Також розглянуто основні історичні події та переваги і недоліки цієї технології.

РОЗДІЛ 2 ЦЕНТРАЛІЗОВАНІ ТА ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ

2.1 Централізована система

Централізовані системи(рис. 2.1) - це системи, які мають єдиний центральний орган. Весь контроль зосереджений у ньому, який забезпечує всі процеси та рішення здійснюються у цьому місці. Централізовані системи мають декілька переваг. Оскільки є лише одна точка повноважень, рішення мають бути швидшими та простішими у реалізації. Кожен орган відповідає за одне завдання. Це запобігає виконанню будь-якого окремого завдання багато разів, тобто гарантує що, не буде дублікатів[7].

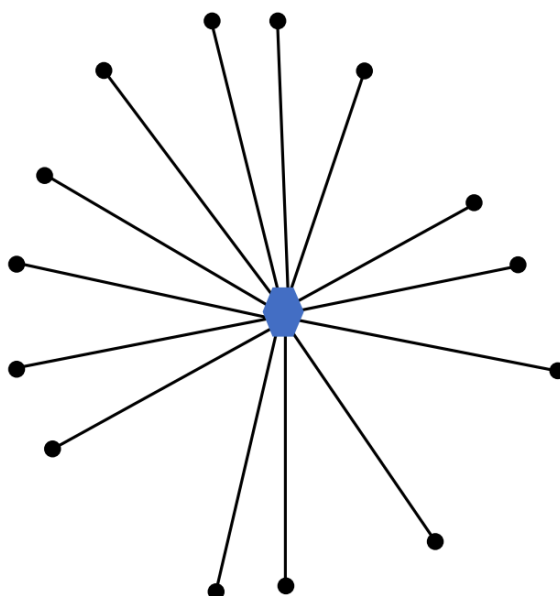


Рисунок 2.1 - централізована система[7]

Централізована система приносить багато переваг і, отже, вона користується популярністю. Однак централізовані системи також мають багато недоліків. Один із головних це наявність єдиного серверу(однієї точки відмови). Наприклад, якщо сервер пошкодиться або його атакують хакери то вся система зруйнується. Ще однією вадою цих систем є те, що централізована система не має прозорості і схильна до шахрайства та недовіри. Централізовані системи працюють як чорний ящик, і люди, які використовують їх, можуть зазнати

впливу шахрайства та втрати довіри. Централізована система має багато переваг і більше підходить для систем, які можуть приймати єдину точку довіри, вимагають швидкого зростання[7].

2.2 Децентралізована система

Децентралізована система(рис. 2.2) не має центрального власника. Натомість використовує декількох центральних власників, кожен із яких зазвичай зберігає копію ресурсів, до яких мають доступ користувачі[8].

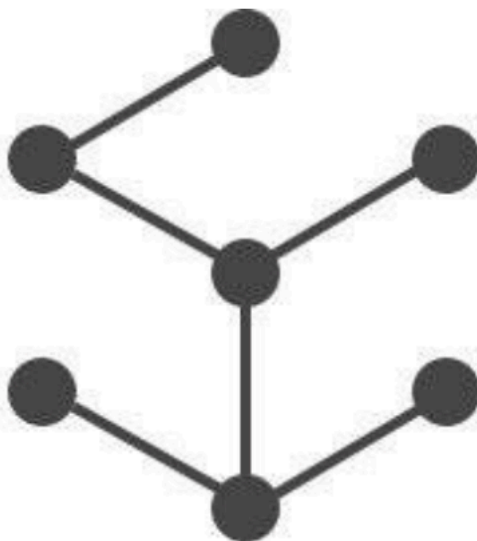


Рисунок 2.2 - децентралізована система[8]

Децентралізована система може бути настільки ж вразливою до збоїв, як і централізована. Це тому, що коли один або кілька центральних власників або серверів виходять з ладу, інші можуть продовжувати надавати доступ до даних користувачам. Ресурси залишаються активними, якщо хоча б один із центральних серверів продовжує працювати. Зазвичай це означає, що власники системи можуть виправити несправні сервери та вирішити будь-які інші проблеми, а сама система продовжує працювати в звичайному режимі[8].

Збої сервера в децентралізованій системі можуть вплинути на продуктивність та обмежити доступ до деяких даних. Але з точки зору загальної тривалості роботи системи ця система пропонує значне покращення порівняно з централізованою системою[8].

Ще однією перевагою є те, що час доступу до даних часто є швидшим. Це тому, що власники можуть створювати вузли в різних регіонах або областях, де активність користувачів висока[8].

Однак децентралізовані системи все ще схильні до тих самих ризиків для безпеки та конфіденційності для користувачів, що і централізовані системи. І хоча їхня відмовостійкість вища, це має свою ціну. Утримання децентралізованої системи, як правило, дорожче[8].

2.3 Приклади

2.3.1 Платіжна система

Очевидною відправною точкою було б говорити про глобальну платіжну систему, оскільки це була початкова ідея першої у світі децентралізованої криптовалюти - біткойн. Кожен банк у світі працює на централізованих серверах. Це означає, що вони мають доступ до всієї фінансової діяльності користувачів[9].

Вони знають, скільки отримуємо коштів та куди їх витрачаємо, кому надсилаємо гроші та все інше, пов'язане з вашим банківським рахунком. Крім того, якби хтось зміг отримати ваш пароль до банку або, що ще гірше, зламати централізовані сервери банку, то отримав би доступ до всієї цієї інформації. Якщо централізовані сервери вийшли з ладу (що відбувається постійно), то користувачеві може бути відмовлено у доступі до його коштів[9].

Коли ми порівнюємо централізовану та децентралізовану, децентралізована платіжна система вирішує всі ці проблеми. Оскільки використовуючи криптовалюту для надсилання або отримання платежів, не потрібно покладатися на третю сторону для підтвердження транзакції. Ось чому

біткойн та деякі інші криптовалюти називають «одноранговими цифровими валютами»[9].

Оскільки для третьої сторони немає вимог, плата значно менша, а в деяких випадках практично безкоштовна. Це робить криптовалюти ідеальними під час надсилання або отримання платежу від когось в іншій країні, оскільки такі компанії, як Western Union, беруть високі суми[9].

Децентралізовані системи не мають кордонів, тому не має значення, кому надсилаються кошти чи то в інше місто чи в другу країну. Це займає однакову кількість часу, а плата однакова.

Кошти також набагато безпечніші при використанні децентралізованої системи. Єдиний хто має доступ до своїх коштів це сам користувач, оскільки тільки він має приватний ключ до свого банківського аккаунту. Якщо дотримуватися правильних заходів безпеки, ніхто не зможе дізнатися, що це за приватні ключі[9].

Загалом, децентралізована глобальна платіжна система має наступні переваги:

- жодна третя сторона не може отримати доступ до вашої інформації.
- швидкі та дешеві транзакції в усьому світі.
- прозорий - кожен може бачити транзакції, що відбулися в мережі.
- немає центральної точки відмови.

2.3.2 Вибори президента

У більшості країн громадянами обирають президентів на чергових виборах. Цей процес повинен бути простим, кожному повинен бути наданий один голос, і він повинен здійснюватися в чесному та прозорому процесі. Однак навіть у західному світі це не завжди так[9].

На останніх виборах в США наприкінці 2016 року Дональд Трамп від Республіканської партії переміг і став новим президентом. Під час виборів кожен громадянин віком від 18 років мав шанс проголосувати за бажаного президента, що є найчеснішим способом проголосувати[9].

Однак незабаром було багато повідомлень у ЗМІ та змов, які заявляли, що голосування було маніпульоване. Було навіть проведено повне розслідування того, чи російський уряд якимось маніпулював процесом. Однак, оскільки громадськість не може отримати доступ до даних про голосування, люди не можуть знати, правда це чи ні[9].

Не лише це, але бували випадки, коли уряди шахрайським шляхом перемагали на виборах, маніпулюючи голосами.

Одним із таких рішень цих загроз є голосування за децентралізовану систему. Ця система є прозорою і дозволить усім виборцям бачити, звідки походить кожен голос, і легітимність кожного голосування. Це гарантувало б, що люди отримують лише один голос, і в систему неможливо втрутитись тим самим змінювати голос[9].

Існує одна криптовалюта VoteCoin, яка дозволить людям голосувати на безпечній та анонімній платформі, гарантуючи, що результат буде справедливим і прозорим. Нажаль, вибори на базі децентралізованої системи ще не відбулися[9].

Переваги децентралізованої системи голосування:

- немає шансів на фальсифікацію виборів чи маніпуляції через прозорість мережі.
- немає можливості погрози з боку урядових сил через анонімність.

2.3.3 Електроенергія

Електроенергія - потрібна усім людям, незалежно від того, чи освітлювати будинок, чи жити плити та просто нагрівати воду. Однак нинішня централізована система - все, крім справедливості[9].

Стандартний процес працює так: приватна централізована організація виступає посередником між громадянином та національною мережею. Національна мережа - це інфраструктура, яка з'єднує будинки людей з електропостачанням[9].

Однак ці централізовані треті сторони обирають, скільки вони стягують з людей, а це означає, що вони отримують величезний прибуток на чомусь, що слід класифікувати як основне право людини. Це не означає, що електроенергія повинна бути безкоштовною, але немає необхідності мати централізовану третю сторону, яка стягує великі кошти[9].

Одним із рішень, яке зараз розробляється, є децентралізована організація під назвою PowerLedger. Використовуючи технологію блокчейн, створили систему, яка виключає посередника[9].

Користувач має у своєму будинку сонячні батареї і використовує потрібну йому електроенергію. У нього залишилася зайва електроенергія, яку він планує продати. Зазвичай йому доведеться продати його назад електроенергетичній компанії, однак за допомогою PowerLedger він може продати комусь іншому безпосередньо за справедливою ринковою ціною. Його запит публікується в загальнодоступному блокчейні, і будь-хто в околицях може придбати у нього додаткову електроенергію. Не тільки користувач отримує кращу ціну, але й покупець[9].

Все це можна зробити без допомоги централізованої третьої сторони. Як видно з наведеного прикладу, виключаючи сторонніх виробників, і покупець, і продавець отримують вигіднішу ціну за свою електроенергію. Ніхто не може обдурити систему, оскільки все доступне для перегляду на блокчейні. Щоб зробити речі ще більш безпечними, покупець і продавець можуть торгувати, використовуючи криптовалюту, що дозволяє здійснити майже миттєву та безкоштовну транзакцію[9].

Переваги:

- справедлива ринкова ціна як для покупця, так і для продавця.
- не потрібно третьої сторони (яка отримує прибуток як від покупця, так і від продавця).
- покупець оплачує криптовалютою, що швидше та дешевше, ніж традиційні способи оплати.

2.4 Висновки до розділу

У цьому розділі йдеться мова про централізовані та децентралізовані системи, у чому їх сутнітність, де використовуються. Також були наведені приклади реальних кейсів, та порівняння між собою централізованих та децентралізованих систем. На мою думку, найкращий випадок де можна використати децентралізацію це вибори президента з розглянутих кейсів. Оскільки вибори це найбільша подія у світі, але досить часто тут шахрують та змінюють голоси людей на чийсь користь. Я вважаю, що децентралізація необхідна тут перш за все. Оскільки все стане прозорим та ніхто не зможе фальсифікувати голоси.

РОЗДІЛ 3 ІНСТРУМЕНТИ БЛОКЧЕЙНУ ДЛЯ ВЕБ-РОЗРОБКИ

3.1 Додатки з використанням блокчейна

Створити власний додаток з використанням блокчейн можливо трьома способами(див. рис. 3.1).

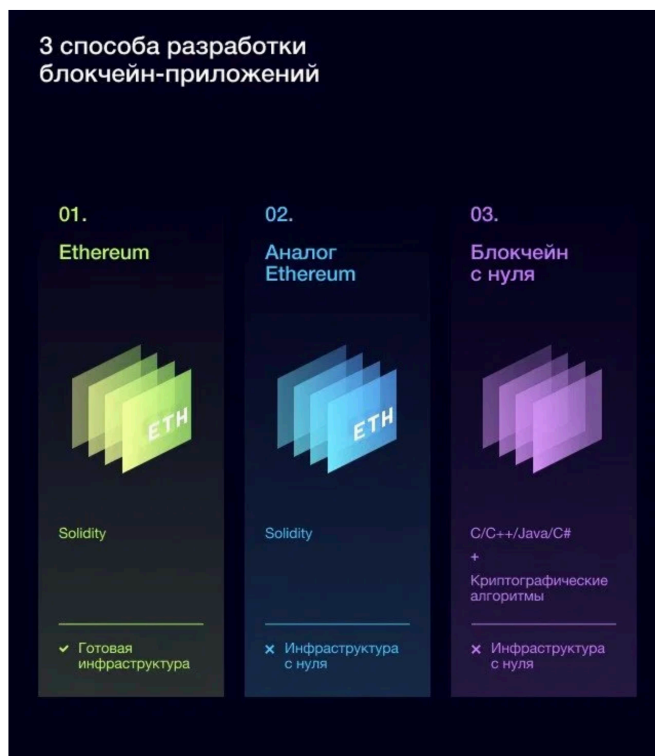


Рисунок 3.1 - шляхи створення блокчейн додатків[10]

3.1.1 Використання платформи для блокчейн-додатків

Створення програми з використанням готового блокчейна на одній із спеціалізованих платформ - це самий простий і швидкий спосіб розробки програми. Найбільш популярні платформи:

- 1) Ethereum - одна з найвідоміших і функціональних блокчейн-платформ з розвиненою інфраструктурою для створення смарт-контрактів і розробки децентралізованих додатків. Основна мова програмування - Solidity, внутрішня криптовалюта - Ether (ETH)[10].

- 2) Hyperledger Fabric - перспективна платформа для розробки рішень на базі блокчейн, орієнтована на корпоративний сегмент. З'явилася за підтримки IBM і в даний момент не має своєї криптовалюта. Актуальні мови програмування: Go, Java, JavaScript[10].
- 3) Cardano - діюча блокчейн-платформа нового типу; ключова відмінність від інших платформ полягає в тому, що в Cardano існує поділ обчислювальних шарів: перший шар призначений для роботи з внутрішньою криптовалютою ADA, другий - для створення і обробки смарт-контрактів, написаних на мові програмування Haskell[10].

Вибір платформи залежить від складності та цілі проекту. Для реалізації блокчейна на платформі Ethereum, необхідно створити новий смарт-контракт на високорівневій мові Solidity, чий синтаксис схожий на JavaScript, і завантажити його в мережу Ethereum, де вже є свій API і все необхідне для швидкого запуску готового продукту. Однак не все так гладко: у даному випадку, використовується блокчейн-платформа, Ethereum, і це означає, що з кожної транзакції і при кожній зміні, внесеному до смарт-контракту, стягується комісія в ЕТН за використання обчислювальних ресурсів платформи, що підтримує роботу і забезпечує роботу мережі. І разом з популярністю додатку і розширенням клієнтської бази буде паралельно зростати рівень витрат[10].

Плюси:

- блокчейн-платформа вимагає мінімум зусиль з боку розробника на початковому етапі і разом з усіма необхідними інструментами для розробки програми надає готову мережу та інфраструктуру зі своїми серверами і майнерами[10].

Мінуси:

- за кожне використання обчислювальних ресурсів платформи стягується комісія в ЕТН. Це означає, що витрати будуть постійно збільшуватися пропорційно зростанню популярності додатку[10].

На розробку ядра блокчейн-додатки з використанням готового блокчейна в середньому йде 2 тижні.

3.1.2 Власна мережа на базі open-source блокчейн-платформи

І тут нам на допомогу знову приходить блокчейн-платформа Ethereum. Ethereum - це платформа з відкритим вихідним кодом, який досвідчений розробник може адаптувати під цілі і потреби конкретного проекту, розгорнувши власну мережу-клон Ethereum, і створити унікальний, незалежний децентралізований продукт. Даний варіант насамперед привабливий своєю автономністю: ваша мережа - ваші правила. За рахунок використання відкритого коду, економиться час і ресурси, необхідне для створення власної мережі[10].

Плюси:

- створення блокчейна на базі open-source платформи звільняє від сплати комісії за кожен проведений транзакцію[12].
- Всі витрати будуть обмежуватися статичної щомісячної сумою на підтримку серверів та інфраструктури і не будуть збільшуватися з ростом транзакцій[10].

Мінуси:

- доведеться самостійно створювати і підтримувати інфраструктуру: від моніторингу і масштабування до настройки обладнання для обслуговування мережі (Майнінг), що передбачає додаткові систематичні витрати[10].

Розробка програми зі створенням власної блокчейн-мережі займає мінімум 1 місяць.

3.1.3 Створення проекту з використанням блокчейна з нуля

Розробка блокчейна з нуля має на увазі створення додатка на одному з високорівневих мов програмування (C, C++, Javascript, Python, C#, Java) з використанням величезної кількості криптографічних алгоритмів. Криптографічні алгоритми - фундаментальна основа блокчейна. Для підпису і верифікації будь-яких даних усередині інформаційного блоку проводяться різні процедури, що складаються з складних арифметичних операцій, що, по суті, переносить розробку блокчейна швидше в область математики, ніж програмування[10].

Плюси:

- відсутність комісії за використання обчислювальних ресурсів платформи[12].
- створення блокчейна з нуля обгрунтовано при наявності унікальних умов або обмежень (наприклад, якщо критично, щоб мережа функціонувала в рамках конкретних держав)[10].
- також у випадках, коли існуючі алгоритми готових блокчейнов не відповідає вашим поставленим цілям і завданням[10].

Мінуси:

- довгий термін розробки.
- так само, як і при створенні власної мережі на базі open-source платформи, розробка блокчейна з нуля вимагає створення повноцінної інфраструктури (включаючи настройку серверів, адміністрування обладнання та підключенням Майнер), яку доведеться моніторити і підтримувати самостійно[10].

Розробка власного блокчейна займає мінімум 6 місяців.

3.2. Інструменти блокчейну для веб розробки

Виникнення блокчейн зосереджувалось насамперед на створенні децентралізованої платіжної системи. Отже, світ вперше побачив криптовалюту в 2009 році, і з тих пір технологія блокчейн пройшла довгий шлях. На сьогоднішній день блокчейн - це не що інше, як великий, децентралізований суперкомп'ютер, який забезпечує ідеальну платформу для розробників по всьому світу для розробки власних програмних додатків[11].

Як результат, попит на найкращі інструменти блокчейн значно зростає. Наступне обговорення ілюструє деякі найкращі інструменти блокчейну, які можуть забезпечити різноманітні функціональні можливості для розвитку блокчейну. Перш ніж викласти інструменти блокчейну, давайте коротко поміркуємо про попит на технологію блокчейну та значення інструментів блокчейну[11].

Звіт Upwork Index of Skills за 2018 рік продемонстрував блокчейн як одну з найбільш швидкозростаючих навичок з точки зору попиту. Крім того, нещодавній звіт LinkedIn поставив блокчейн серед провідних технологічних навичок, що користуються попитом у 2021 році. Технологія блокчейн - це постійно розвивається простір з величезним потенціалом для інновацій[11].

Таким чином, попит на спеціалістів із блокчейнів, які мають досвід роботи з найкращими інструментами блокчейну, в довгостроковій перспективі значно збільшиться. Існує безліч кар'єрних ролей, які можна дослідити в блокчейн. Однак потрібно дізнатися про різні інструменти розробників блокчейнів та навички їх реалізації для різних випадків використання[11].

Основною функцією інструментів блокчейну є задоволення різних вимог на різних етапах розробки додатків блокчейну. Інструменти блокчейну відповідають за спрощення процесу розробки програмних рішень блокчейну[11].

3.2.1 Geth

Одне з перших згадувань серед найкращих доступних на даний момент інструментів блокчейну - Geth. В основному це реалізація вузла Ethereum, заснована на мові програмування Go. Можна скористатися Geth у трьох різних інтерфейсах, таких як інтерактивна консоль, сервер JSON-RPC та командний рядок. Geth може підтримувати розробку блокчейнів в операційних системах Linux, Windows та Mac[11].

Він служить ідеальним інструментом для широкого кола завдань Ethereum Blockchain, включаючи створення смарт-контрактів, передачу токенів, вивчення історії блоків та видобуток токенів ефіру. Після встановлення Geth користувачі можуть підключитися до існуючого блокчейну або розробити власний. Фактором переваги, в даному випадку, є здатність Geth спрощувати ситуацію за допомогою автоматичного з'єднання з основною мережею Ethereum[11].

3.2.2 Remix IDE

Remix IDE - одне з найпоширеніших імен, що входить до числа найкращих інструментів блокчейну на 2020 рік. Це інструмент блокчейну, орієнтований на браузер, який підтримує процеси розробки та розгортання смарт-контрактів. Remix IDE базується на JavaScript, тим самим надаючи доступ через будь-який сучасний браузер[11].

Крім того, Remix також може допомогти у вирішенні функцій написання смарт-контрактів у Solidity та тестуванні, налагодженні та розгортанні їх. Користувачі також можуть віддавати перевагу використовувати Remix локально або через браузер, залежно від їхніх уподобань. Найголовніше за все, Remix надає вичерпну та просту документацію для полегшення безперебійного підключення до блокчейну Ethereum за допомогою Metamask[11].

3.2.3 Mist

Якщо потрібен гаманець Ethereum на існуючому ринку, то Mist є однією з найкращих альтернатив. Вражаючим фактором Mist, який вводить його серед найкращих інструментів блокчейну, є те, що Ethereum з ним. Перед використанням Mist користувачі повинні мати відведене місце для зберігання токенів Ether та виконання смарт-контрактів[11].

Mist підтримує як 32-, так і 64-розрядні версії Windows, 32- і 64-розрядні версії операційних систем Linux і Mac. В даний час Mist служить ідеальним вибором для розгортання смарт-контрактів, а також є гаманцем на повних вузлах. Отже, користувачам доведеться завантажити повний блокчейн Ethereum, розмір якого перевищує 1 ТБ. Крім того, користувачі повинні пам'ятати свій пароль Mist завжди, оскільки вони ніколи не можуть його змінити[11].

3.2.4 Truffle

Одним із помітних конкурентів серед найкращих інструментів блокчейну є компанія Truffle. Це фреймворк Ethereum blockchain, розроблений для створення середовища розробки для полегшення розробки рішень на основі Ethereum. Truffle також включає величезну бібліотеку спеціальних розгортань, які підтримують написання нових смарт-контрактів та вирішення проблем у розробці блокчейну[11].

Крім того, Truffle також ідеально підходить для розробки складних децентралізованих програм Ethereum. Ще однією визначною функцією Truffle як інструменту блокчейну є автоматизоване тестування контрактів. Трюфель може використовувати Mocha та Chai для автоматизованого тестування контракту. Крім того, Truffle може також допомогти у розробці смарт-контрактів з подальшим їх зв'язком, їх компіляцією та розгортанням. Найважливіше з усіх, Truffle також сприяє налаштуванню конвеєру збірки, щоб забезпечити виконання спеціальних процедур збірки[11].

3.2.5 Metamask

В даний час прийняття технології блокчейн значною мірою обертається навколо криптовалют. Отже, гаманці також вважаються найкращими інструментами блокчейну, і Metamask має деякі унікальні функціональні можливості, крім цього. Він служить зв'язком між браузером та блокчейном Ethereum[11].

Metamask також надає програмну платформу для обслуговування ефіру або інших активів ERC-20. У той же час це також допомагає взаємодіяти з децентралізованими програмами Ethereum. Функціонал, що базується на браузері, забезпечує потенційне підвищення зручності використання Metamask. Важливо знати, що користувачі можуть пов'язувати Coinbase та Shapeshift з Metamask для продажу та купівлі токена ERC20 та ETH[11].

Користувачі також можуть використовувати Metamask для збереження ключів для токенів ETH та ERC20. Metamask має можливість взаємодії з різними тестовими мережами Ethereum, що робить його відповідним гаманцем для розробників блокчейнів. Після встановлення програми в браузері користувачі можуть мати у своєму розпорядженні вбудований гаманець Ethereum[11].

3.2.6 Parity

Parity насправді є клієнтом для Ethereum. Це ще один популярний інструмент серед розробників блокчейнів, оскільки це інструмент управління, який пропонує критично важливе використання. Крім того, Parity використовує мову програмування Rust. Ще однією чудовою особливістю цього інструменту є те, що він забезпечує інфраструктуру, яка може похвалитися швидким та надійним сервісом[11].

У будь-якому випадку, побудувати власну мережу блокчейнів для приватного використання чи підприємств або навіть просто для дослідницьких цілей надзвичайно просто.

Основні переваги Parity:

- Швидші транзакції;
- Високодосконалий клієнт на основі CLI;
- Різні підключаються алгоритми консенсусу;
- Малий обсяг пам'яті та зменшений обсяг пам'яті;
- Warp Sync допомагає швидше синхронізуватися;
- Вищий контроль за конфіденційністю та доступом;
- Модульний дизайн та налаштування;

3.2.7 Ganache

Як інструмент Blockchain від Truffle Suite, Ganache дозволяє розробникам Blockchain створювати власний приватний блокчейн Ethereum для тестування dapps, перевірки стану та виконання команд, забезпечуючи повний контроль роботи ланцюжка[11].

Найважливішою характеристикою Ganache є те, що він дозволяє користувачам виконувати всі дії, які можна було б виконати в основному ланцюгу, не сплачуючи за це. Ganache - це дуже популярний інструмент Blockchain серед розробників, оскільки він пропонує ряд опцій, таких як вбудований провідник блоків та вдосконалений контроль майнінгу. Розробники блокчейнів використовують Ganache для тестування своїх смарт-контрактів у процесі розробки[11].

3.2.8 Embark

Embark, будучи фреймворком для Ethereum dapps, дозволяє розробникам легко виконувати розробку та розгортання dapps. Це також дозволяє користувачам розробляти та розгортати безсерверну програму html5, яка використовує децентралізовані технології. Можна також створити смарт-контракти за допомогою програми Embark і зробити їх доступними в коді JS.

Якщо контракт буде змінено, Embark змінить його самостійно разом із відповідним додатком[11].

Міграція смарт-контрактів, передислокація контрактів та зміни можуть бути легко оброблені у випадку кількох контрактів. Для взаємодії з Embark можна використовувати традиційну мову веб-розробки та будь-який конвертер збірки. Embark підтримує розвиток контрактів з Javascript на тестовій основі, що дозволяє розробникам управляти своїми контрактами на різних блокчейнах[11].

3.3 Висновки до розділу

У цьому розділі було показано три способи створення блокчейн додатків. Перший це коли маємо готовий блокчейн, наприклад Ethereum, в який надає готову мережу та інфраструктуру зі своїми серверами. Потрібно тільки завантажити смарт-контракт, написаний на Solidity, в мережу Ethereum, але при кожній транзакції та зміні смарт-контракту буде стягуватися комісія в ЕТН за використання обчислювальних ресурсів платформи.

Другий це використання open-source блокчейн-платформи для створення власної мережі. Наприклад Ethereum, оскільки в нього відкритий вихідний код. За рахунок використання відкритого коду, економиться час і ресурси, необхідні для створення власної мережі. Також комісія за транзакції відсутня, але необхідно самому створити та підтримувати інфраструктуру.

Третій це створення додатку з використанням блокчейн з нуля. У цьому випадку створюється блокчейн за допомогою криптографічних алгоритмів(підпис, верифікація інформації, транзакцій і так далі). При створенні блокчейну з нуля можна його адаптувати під цілі проекту, що зручно, проте розробка такого додатку тривати до шести або більше місяців.

Також було розглянуто інструменти блокчейну для веб-розробки. У наступному розділі декілька з них використаю для реалізації веб-додатку.

РОЗДІЛ 4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Веб-додаток Todo-List

Буде реалізовано веб-додаток Todo-List двома способами централізовано та децентралізовано.

Стек технологій для централізованого:

- Vue(фронтенд фреймворк)
- Firebase(база даних)

Стек технологій для децентралізованого:

- Vue
- Truffle
- Metamask
- Ganache
- Solidity

Функціонал у обох додатків буде однаковий, а саме:

- створення нового завдання
- видалення завдання
- відмітка про те, що завдання виконане
- оновлення завдання
- видалення завдання

4.2 Технології для централізованого додатку

4.2.1 Vue

Vue це Фронтенд-фреймворк для написання інтерфейсів користувача. Вибрав його, оскільки є детально описана документація(не лише описується, що за “функція”, але й наведені приклади використання, та де краще застосовувати), тому він простий в освоєванні.

4.2.2 Firebase

Firebase це хмарна база даних, яка дозволяє користувачу робити основні CRUD операції над даним, та має зручні методи взаємодії з нею. Вибрав її, оскільки мені не потрібно створювати базу даних, писати арі для отримання та видачі даних. Всю серверну частину виконує firebase.

4.3 Технології для децентралізованого додатку

4.3.1 Truffle

Truffle надає набір інструментів для роботи зі смарт-контрактами. Управління смарт-контрактами - створення смарт-контрактів на мові програмування Solidity та компілювання їх до байт-коду, який запускатиметься на віртуальній машині Ethereum (EVM)[12].

Автоматизоване тестування - написання тестів для смарт-контрактів, щоб переконатися, що вони поведуться так, як потрібно. Ці тести можуть бути написані на JavaScript або Solidity і можуть бути запуснені на будь-якій мережі, налаштованій Truffle, включаючи загальнодоступні мережі блокчейнів[12].

Розгортання та міграція - написання сценаріїв для міграції та розгортання смарт-контрактів у будь-якій загальнодоступній мережі блокчейнів Ethereum[12].

Керування мережею - підключення до будь-якої загальнодоступної мережі блокчейнів Ethereum, а також до будь-якої особистої мережі блокчейнів, яку можна використовувати для цілей розробки[12].

Консоль розробки - взаємодіяти зі смарт-контрактами в середовищі виконання JavaScript за допомогою консолі Truffle. Для цього можна підключитися до будь-якої мережі блокчейнів, яка вказана у конфігурації мережі[12].

Script Runner - написання власних скриптів, які можуть працювати на загальнодоступній мережі блокчейнів за допомогою JavaScript. Можна написати

будь-який довільний код всередині цього файлу та запустити його у своєму проєкті[12].

Розробка на стороні клієнта - налаштування свого проєкту Truffle для розміщення програм на стороні клієнта, які розмовляють із вашими смарт-контрактами, розгорнутими на блокчейні[12].

4.3.2 Metamask

Якщо не використовувати Metamask, то доведеться завантажити клієнт Ethereum і синхронізувати весь ланцюжок блоків на локальний комп'ютер, тобто наш компютер буде вузлом. Це займе багато місця на жорсткому диску та витрачатиме ресурси мережі на синхронізацію з мережею. Також, в Dapp потрібно вручну вбудовувати і налаштовувати під необхідну ноду Web3.js[13].

Metamask вирішує ці проблеми і полегшує роботу з блокчейном мережі Ethereum. Metamask - це розширення для браузера Chrome та Firefox, яке є "мостом" між браузером і блокчейном Ethereum. При роботі з основним блокчейном і тестовими, Metamask надає доступ до своїх віддалених вузлів, тим самим позбавляє від необхідності завантажувати весь ланцюжок блоків потрібного блокчейна. Ще однією особливістю Metamask є автоматична інжекція об'єкта Web3js в усі веб-сторінки. Даний об'єкт вже налаштований на мережу і аккаунт, які обрані в інтерфейсі розширення[13].

Переваги:

- популярний, зазвичай використовується як плагін для Dapp
- простий, замість управління приватними ключами потрібно запам'ятати список слів і транзакції підписуються від їх імені
- економить місце на диску, не потрібно завантажувати блокчейн Ethereum, оскільки він надсилає запити на вузли поза комп'ютером користувача

Недоліки:

- Metamask зберігає приватні ключі в браузері користувача. Це менш безпечно, ніж апаратний або паперовий гаманець. Це зручний компроміс для простоти використання.
- зовнішні вузли, замість того, щоб бути повноцінним вузлом, він покладається на зовнішні вузли, які іноді мають час простою, що може призвести до того, що Metamask перестане працювати.

4.3.3 Ganache

Для плати за транзакцію нам необхідний gas, який конвертується в ЕТН. Для цього потрібно мати якийсь гаманець з ефірами, Ganache для цього ідеально підходить. Ganache - це персональний блокчейн для створення розподілених додатків за допомогою Ethereum. Він дає 10 безкоштовних акаунтів зі 100 підробленими ефірами кожен для цілей тестування[12].

4.3.4 Solidity

Solidity - це об'єктно-орієнтована мова програмування високого рівня для написання смарт-контрактів в Ethereum. Будь-який код, написаний на Solidity, може бути виконаний на EVM після компіляції в байт-код, який представляє собою набір інструкцій для EVM. Байт-код кодує посилання на інші функції та контракти, які викликаються під час виконання. Це робиться за допомогою двійкового інтерфейсу програми (ABI). ABI є інтерфейс між двома програмними модулями: інструкції на машинному рівні та високорівневою мовою програмування[14]. Тобто:

- контракт це код високого рівня, визначений формальною мовою, такою як Solidity
- скомпільований контракт перетворюється в байт-код для виконання на EVM. Назви функцій та їх вхідні параметри хешуються під час компіляції[15].

- ABI - це перелік назви і аргументів функції контракту в JavaScript (JSON). Назва функції та вхідні аргументи є хешовані в ABI. Це включається в дані транзакції та інтерпретується за EVM на вибраному акаунті. ABI необхідний, щоб визначити яку функцію викликати з контракту, також, щоб мати гарантію, що функція поверне дані у визначеному форматі.[14]

4.4 Архітектура децентралізованого додатку

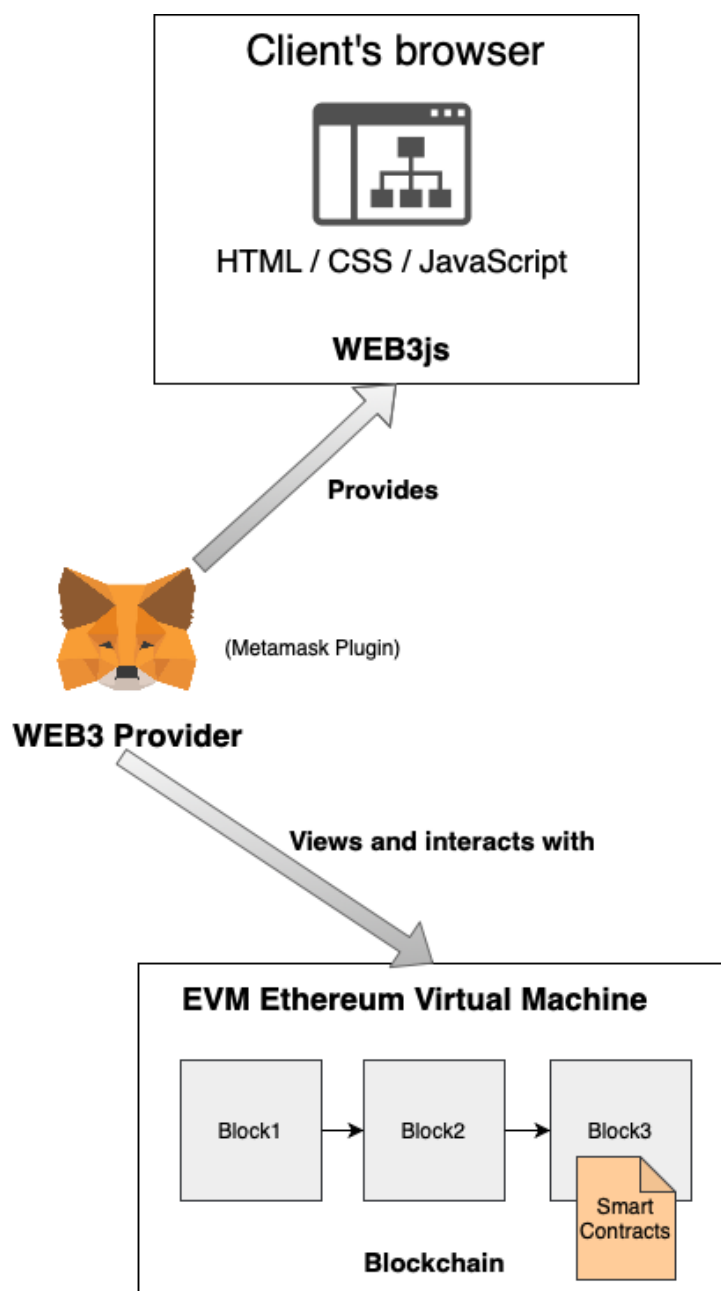


Рисунок 4.1 - архітектура децентралізованого додатку

Архітектура(рис. 4.1) трохи складніша, оскільки на будь-які дії користувача потрібно через Metamask підтверджувати транзакцію, але є комісія за неї, щоб її заплатити береться аккаунт із Ganache і підключається в Metamask після сплати комісії, на Ethereum блокчейн створиться новий блок з цією транзакцією.

4.4.1 Web3js та Web3 Provider

Web3.js - це колекція бібліотек, що дозволяє браузеру взаємодіяти з блокчейном. Це дозволяє читати та записувати дані зі смарт-контрактів, передавати ефіри між обліковими записами та багато іншого[15].

Мережа Ethereum містить вузли, і всі вузли мають однакову копію даних. Встановлення "Web3 Provider" у web3.js повідомляє нашому коду, з якого вузла ми будемо читати та записувати дані. У нашій реалізації ми використовуємо Metamask, яка вводить свого постачальника web3 у браузер[15].

4.4.2 Смарт-контракт

Смарт-контракти - це просто програми, що зберігаються на блокчейні, і працюють, коли виконуються заздалегідь визначені умови. Зазвичай вони використовуються для автоматизації виконання угоди, щоб усі учасники могли негайно бути впевненими в результаті, без участі посередника або втрати часу. Вони також можуть автоматизувати робочий процес, запускаючи наступну дію, коли умови виконуються[16].

Смарт-контракти працюють, дотримуючись простих тверджень "якщо / коли ... тоді ...", які записуються в код на блокчейні. Мережа комп'ютерів виконує дії, коли задані умови виконуються та перевіряються. Ці дії можуть включати передачу коштів відповідним сторонам, реєстрацію транспортного засобу, надсилання повідомлень або видачу квитка. Потім блокчейн оновлюється після завершення транзакції. Це означає, що транзакцію змінити не можна, і результати можуть бачити лише сторони, яким надано дозвіл[16].

4.5 Архітектура централізованого додатку

Архітектура(рис. 4.2) досить проста, Firebase база даних, Vue для написання користувачського інтерфейсу.

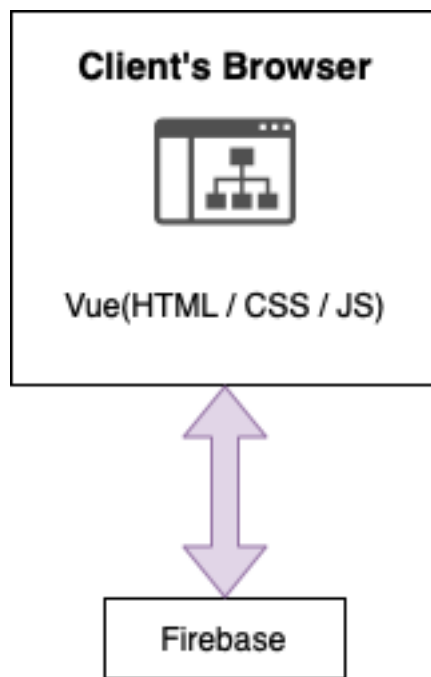


Рисунок 4.2 - архітектура централізованого додатку

4.6 Налаштування середовища для децентралізованого

4.6.1 Установка Truffle

Для початку потрібно встановити Truffle, робимо це наступним чином, відкриваємо термінал та пишемо команду “npm install truffle”, після цього потрібно ініціалізувати його командою “truffle init”(рис. 4.3).

```
MacBook::Vadim:truffle init
✓ Preparing to download
✓ Downloading
✓ Cleaning up temporary files
✓ Setting up box
Unbox successful. Sweet!
Commands:
  Compile:      truffle compile
  Migrate:      truffle migrate
  Test contracts: truffle test
MacBook::Vadim:
```

The image shows a terminal window with a dark background and green text. It displays the output of the 'truffle init' command, which includes a list of steps with checkmarks (Preparing to download, Downloading, Cleaning up temporary files, Setting up box) and a confirmation message 'Unbox successful. Sweet!'. Below this, it lists the available commands: 'Compile: truffle compile', 'Migrate: truffle migrate', and 'Test contracts: truffle test'. The prompt 'MacBook::Vadim:' is visible at the bottom.

Рисунок 4.3 - архітектура централізованого додатку

Після цього папка, куди був завантажений truffle виглядає наступним чином(рис. 4.4).

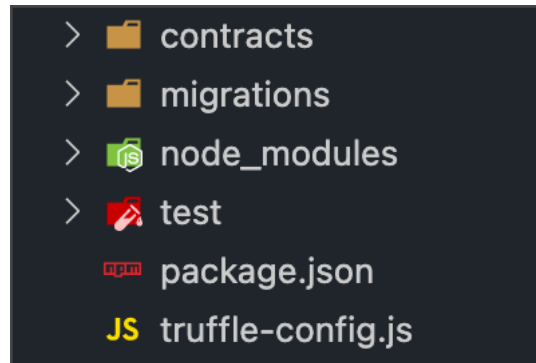


Рисунок 4.4 - завантажені файли

Папка *contracts* - в ній будуть знаходитись файли з розширенням .sol, це смарт-контракти які написані на мові програмування Solidity.

Папка *migrations* - в ній будуть знаходитись файли з розширенням .js, в яких буде описуватися сценарій для деплою контрактів.

Папка node-modules - в ній зберігаються скачені бібліотеки.

Папка *test* - в ній будуть знаходитись файли з розширенням .js, в яких будуть описані тестові сценарії для тестування смарт-контрактів.

Файл *package.json* - в ньому зберігаються назви бібліотек які були скачені через команду “npm install назва_бібліотеки”.

Файл *truffle-config.js* - в ньому прописуються налаштування для роботи truffle.

Розпочнемо з налаштування *truffle-config.js*(рис. 4.5).

```
module.exports = {  
  networks: {  
    development: {  
      host: "127.0.0.1",  
      port: 7545,  
      network_id: "*" // Match any network id  
    }  
  },  
  solc: {  
    optimizer: {  
      enabled: true,  
      runs: 200  
    }  
  }  
}
```

Рисунок 4.5 - *truffle-config.js*

Далі перейдемо до папки contracts, створимо в ній файл todoList.sol та напишемо смарт-контракт(рис. 4.6).

```

1  pragma solidity ^0.5.0;
2
3  contract todoList {
4      uint public taskCount = 0;
5
6      struct Task {
7          uint id;
8          string content;
9          bool completed;
10     }
11
12     mapping(uint => Task) public tasks;
13
14     event Taskcreated(
15         uint id,
16         string content,
17         bool completed
18     );
19
20     event TaskCompleted(
21         uint id,
22         bool completed
23     );
24
25     event TaskUpdated(
26         uint id,
27         string content
28     );
29
30     event TaskDeleted(
31         uint id
32     );
33
34     constructor() public {
35         createTask('test-1');
36     }
37
38     function createTask(string memory _content) public {
39         taskCount = taskCount + 1;
40         tasks[taskCount] = Task(taskCount, _content, false);
41         emit Taskcreated(taskCount, _content, false);
42     }
43
44     function toggleCompleted(uint _id) public {
45         Task memory _task = tasks[_id];
46         _task.completed = !_task.completed;
47         tasks[_id] = _task;
48         emit TaskCompleted(_id, tasks[_id].completed);
49     }
50
51     function updateTask(uint _id, string memory _content) public {
52         Task memory _task = tasks[_id];
53         _task.content = _content;
54         tasks[_id] = _task;
55         emit TaskUpdated(_id, _task.content);
56     }
57
58     function deleteTask(uint _id) public {
59         delete tasks[_id];
60         for (uint i = _id; i < taskCount; i++) {
61             tasks[i] = tasks[i + 1];
62         }
63         taskCount--;
64         emit TaskDeleted(_id);
65     }
66 }

```

Рисунок 4.6 - смарт-контракт

pragma solidity ^0.5.0 - тут вказуємо яку версію Solidity використовувати для компіляції смарт-контракту.

contract todoList{} - це наш смарт-контракт в ньому буду написані необхідні функції.

uint public taskCount = 0 - вказуємо кількість завдань спочатку воно дорівнює 0, але коли користувач буде створювати нові завдання або видаляти їх то ця змінна буде змінюватись, проте меншою за нуль ніколи не буде через *uint*.

struct Task { uint id; string content; bool completed; } - описуємо структуру нашого завдання, тобто *id* буде визначати порядковий номер, *content* це назва завдання, *completed* означає виконано воно чи ні.

mapping(uint => Task) public tasks - це масив елементів типу *Task* та називається він *tasks*.

function createTask(string memory _content) - функція яка створює нове завдання, в аргументах вона приймає назву завдання.

function toggleCompleted(uint _id) - функція яка змінює статус виконання завдання, в аргументах вона приймає порядковий номер завдання.

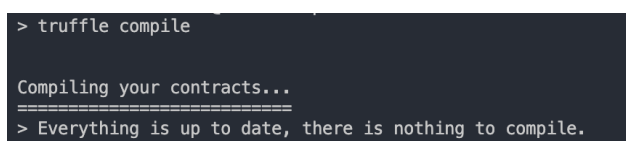
function updateTask(uint _id, string memory _content) - функція яка змінює назву завдання, в аргументах вона приймає порядковий номер завдання та нову назву завдання.

function deleteTask(uint _id) - функція яка видаляє завдання, в аргументах вона приймає порядковий номер завдання.

constructor() public { createTask('test-1') } - конструктор нашого контракту, коли створюється екземпляр контракту, то спрацює конструктор та запустить функцію *createTask('test-1')*.

Також є чотири івента(*event*). При виклику функцій будуть викликатися івенти та зберігати аргументи передані в журналах транзакцій. Ці журнали зберігаються на блокчейні і доступні за адресою контракту.

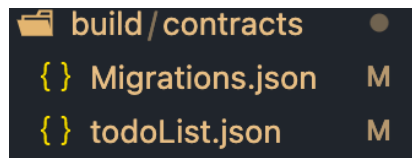
Щоб скомпілювати смарт-контракт необхідно написати команду *truffle compile*(рис. 4.7), після цього створиться папка *build*(рис. 4.8) з необхідними файлами в яких буде інформацією для деплоя контрактів.



```
> truffle compile

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.
```

Рисунок 4.7 - компіляція контракту

Рисунок 4.8 - папка *build*

Далі перейдемо в папку *migrations* та створимо файл *1_initial_migration.js* та напишемо міграцію(рис. 4.9).

```
1  const todoList = artifacts.require("./todoList.sol");
2
3  module.exports = function(deployer) {
4    deployer.deploy(todoList);
5  };
6
```

Рисунок 4.9 - міграція

В даному випадку завантажується контракт *todoList* і за допомогою модуля *deployer* деплоїться в блокчейн. Запуск міграцій здійснюється командою *truffle migrate --reset*(рис. 4.10).

```
> truffle migrate --reset

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name:    'development'
> Network id:     5777
> Block gas limit: 6721975 (0x6691b7)

1_initial_migration.js
=====

Replacing 'todoList'
> transaction hash: 0x5ab28e97f97cea6b21ba62a921b8b3f075677a4b4aed986e28d9a392bbbae66b
> Blocks: 0
> contract address: 0xaeBB2479AC311d543c954262097B0D77bcF3698
> block number: 487
> block timestamp: 1623057000
> account: 0xdE0779a85A294336928825F048D56e96a0dA8282
> balance: 98.88547236
> gas used: 700925 (0xab1fd)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.0140185 ETH

> Saving artifacts
=====
> Total cost: 0.0140185 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.0140185 ETH
```

Рисунок 4.10 - результат команди

Тестування смарт-контрактів перейдемо в папку *test* та створимо файл *todoList.test.js*(рис. 4.11).

```

1  const TodoList = artifacts.require('./TodoList.sol')
2
3  contract('TodoList', () => {
4    before(async () => {
5      this.todoList = await TodoList.deployed()
6    })
7
8    it('deploys successfully', async () => {
9      const address = await this.todoList.address;
10
11      assert.notEqual(address, 0x0)
12      assert.notEqual(address, '')
13      assert.notEqual(address, null)
14      assert.notEqual(address, undefined)
15    })
16
17    it('lists tasks', async () => {
18      const taskCount = await this.todoList.taskCount();
19      const task = await this.todoList.tasks(taskCount);
20
21      assert.equal(task.id.toNumber(), 1)
22      assert.equal(task.content, 'test-1')
23      assert.equal(task.completed, false)
24      assert.equal(taskCount.toNumber(), 1)
25    })
26
27    it('creates tasks', async () => {
28      const result = await this.todoList.createTask('A new task')
29      const taskCount = await this.todoList.taskCount();
30      const event = result.logs[0].args;
31
32      assert.equal(event.id.toNumber(), 2);
33      assert.equal(event.content, 'A new task');
34      assert.equal(event.completed, false);
35      assert.equal(taskCount.toNumber(), 2);
36    })
37
38    it('toggles task completion', async () => {
39      const result = await this.todoList.toggleCompleted(1)
40      const task = await this.todoList.tasks(1)
41      const event = result.logs[0].args;
42
43      assert.equal(task.id.toNumber(), 1)
44      assert.equal(task.completed, true)
45      assert.equal(event.id.toNumber(), 1)
46      assert.equal(event.completed, true)
47    });
48
49    it('update task content', async () => {
50      const result = await this.todoList.updateTask(1, 'updated')
51      const task = await this.todoList.tasks(1)
52      const event = result.logs[0].args;
53
54      assert.equal(task.id.toNumber(), 1)
55      assert.equal(task.content, 'updated')
56      assert.equal(event.id.toNumber(), 1)
57      assert.equal(event.content, 'updated')
58    });
59
60    it('delete task', async () => {
61      const result = await this.todoList.deleteTask(2)
62      const taskCount = await this.todoList.taskCount()
63      const event = result.logs[0].args;
64
65      assert.equal(taskCount.toNumber(), 1)
66      assert.equal(event.id.toNumber(), 2)
67    });
68
69  })

```

Рисунок 4.11 - тести

Буде шість тестів(рис. 4.12):

- чи правильно проходить деплой смарт-контракту на блокчейн
- чи правильно рахує кількість завдань
- чи створює нове завдання
- чи змінює статус завдання

- чи змінює назву завдання
- чи видаляє завдання

```
Contract: TodoList
✓ deploys successfully
✓ lists tasks (234ms)
✓ creates tasks (133ms)
✓ toggles task completion (198ms)
✓ update task content (244ms)
✓ delete task (228ms)

6 passing (1s)
```

Рисунок 4.12 - всі тести успішно пройдено

На даному етапі смарт-контракти компілюються без помилок та роблять деплой на блокчейн.

4.6.2 Установка Metamask

Metamask це розширення для Chrome. У браузері пишемо *chrome://extensions/* та завантажуюємо metamask(рис. 4.13).

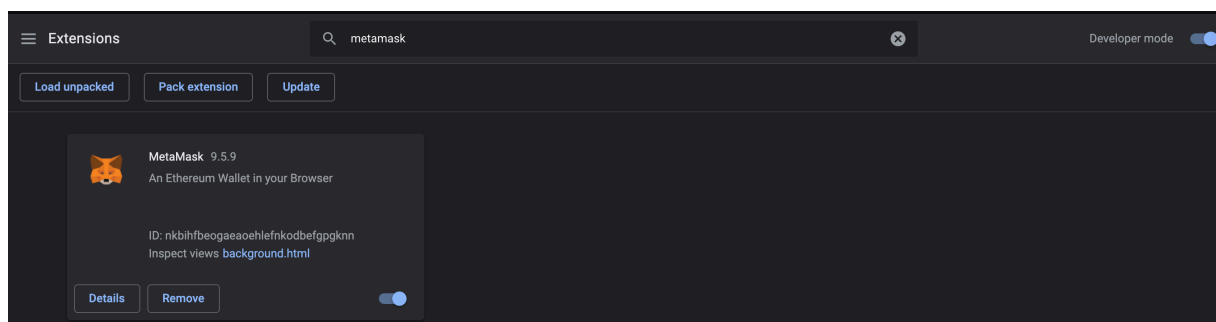


Рисунок 4.13 - завантаження metamask

Після цього завантажуюємо Ganache та відкриваємо його(рис. 4.14).

CURRENT BLOCK 492	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE CARELESS-VEGETABLE	SWITCH	⚙️
MNEMONIC ? merry hard juice web awesome nation rifle course urge file bargain answer							HD PATH m/44'/60'/0'/0/account_index		
ADDRESS 0xDE0779a85A294336928825F048D56e96a0dA8282		BALANCE 98.87 ETH		TX COUNT 357		INDEX 0		🔑	
ADDRESS 0x77Cb6a8D8E045C40Cf270a6f70636F504e3ad57d		BALANCE 99.88 ETH		TX COUNT 135		INDEX 1		🔑	

Рисунок 4.14 - Ganache

Далі натискаємо на ключ відкривається наступне вікно(рис. 4.15), та копіюємо приватний ключ. Переходимо в metamask натискаємо з права наверху *круг* та вибираємо *імпортувати рахунок*(рис. 4.16). В новому вікні вставляємо приватний ключ.

ACCOUNT INFORMATION

ACCOUNT ADDRESS
0xDE0779a85A294336928825F048D56e96a0dA8282

PRIVATE KEY
fed14e627815690351cadbfef9f9bfa547493912bae0b168d9e21a3028af80ff8
Do not use this private key on a public blockchain; use it for development purposes only!

DONE

Рисунок 4.15 - приватний ключ

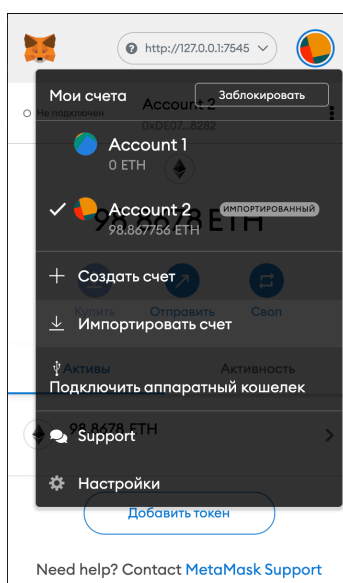


Рисунок 4.16 - імпорт рахунку

Рахунок додано, тепер необхідно налаштувати мережу. Переходимо в настройки та вибираємо пункт мережі(рис. 4.17).

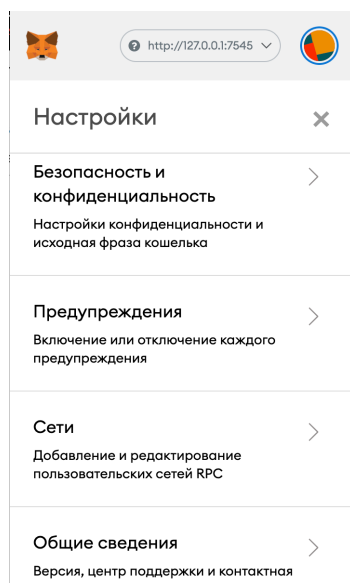


Рисунок 4.17 - мережі(сети)

Добавляємо нову мережу з такими налаштуваннями(рис. 4.18). Тепер Metamask налаштований та смарт-контракти працюють залишається встановити Vue.

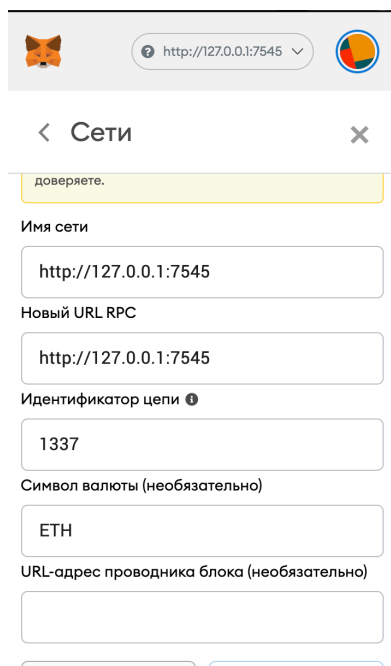


Рисунок 4.18 - налаштування мережі(сети)

4.6.3 Установка Vue

Встановлюємо Vue через команду *npm install vue*. Наступна команда *vue create todo-list*. Прісет вибираємо *Default (Vue 3 Preview) ([Vue 3] babel, eslint)* (рис. 4.19).

```
MacBook::Vadim:vue create todo-list

Vue CLI v4.5.10

New version available 4.5.10 → 4.5.13
Run npm i -g @vue/cli to update!

? Please pick a preset:
  Vue3_Template ([Vue 3] node-sass, babel, router, eslint)
  Vue3_ALL ([Vue 3] node-sass, babel, router, vuex, eslint)
  Default ([Vue 2] babel, eslint)
> Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
  test.confog ([Vue 2] node-sass, babel, pwa, router, vuex, eslint)
  tess ([Vue 2] node-sass, babel, router)
(Move up and down to reveal more choices)
```

Рисунок 4.19 - налаштування Vue 3

Директорія має виглядати наступним чином(рис. 4.20). На цьому етапі установка завершена, залишається написати інтерфейс програми.

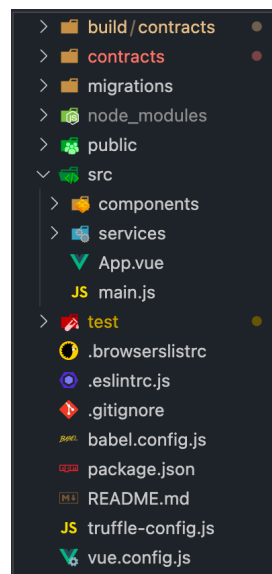


Рисунок 4.20 - директорія

4.7 Налаштування середовища для централізованого

Оскільки, встановлення Vue вже було продемонстровано раніше, то тут сконцентруємось на встановленні Firebase.

4.7.1 Установка Firebase

Для цього потрібно мати аккаунт в Google, після заходимо на firebase.google.com після переходимо в консоль(рис. 4.21), розташовано з права.

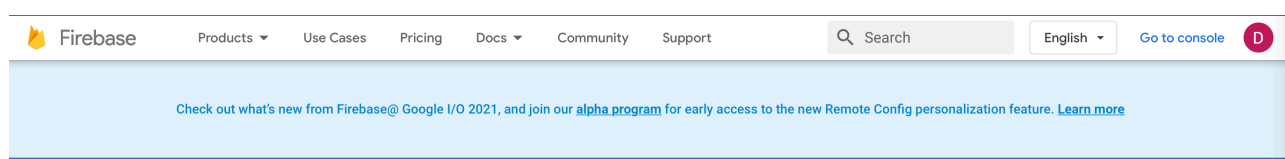


Рисунок 4.21 - firebase

Після створюємо новий проект(рис. 4.22). У новому вікні пишемо назву нашого проекту(рис. 4.23) та переходимо далі.

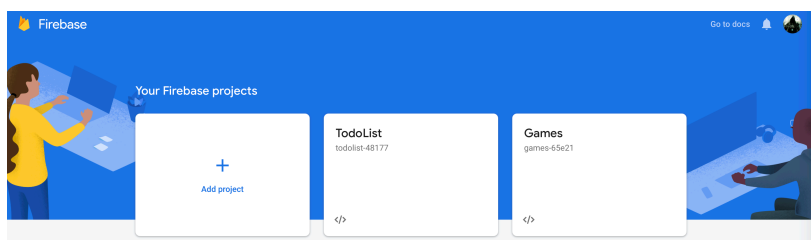


Рисунок 4.21 - новий проект

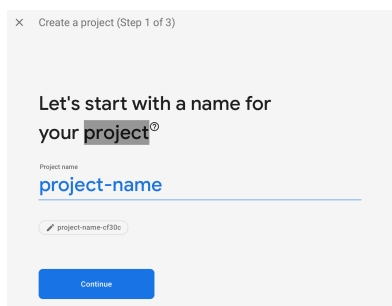


Рисунок 4.22 - назва проекту

У новому вікні(рис. 4.23) можна включити Google Analytics, це за бажанням, якщо включити то у новому вікні потрібно вибрати аккаунт для аналітики або створити. Після цього буде кнопка створити проект.

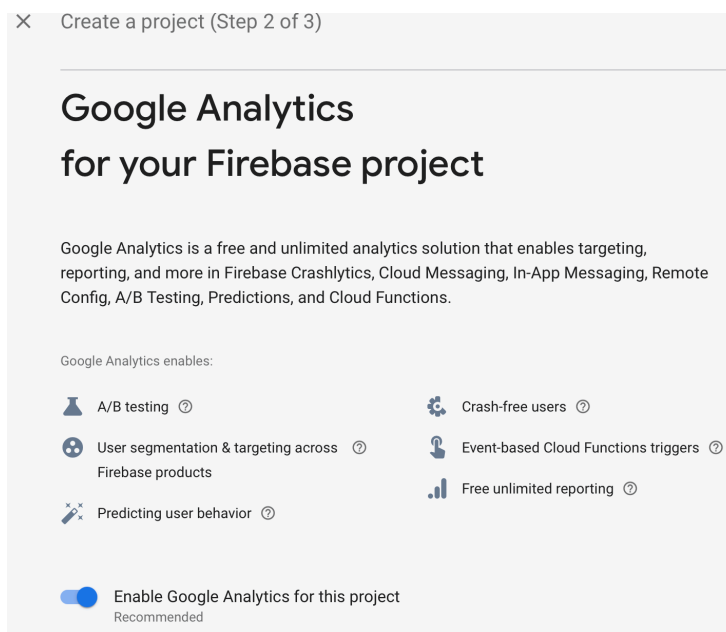


Рисунок 4.23 - Google Analytics

Після створення проекту потрібно натиснути на значок веб-додаток(рис. 4.24), який знаходиться між android та unity.

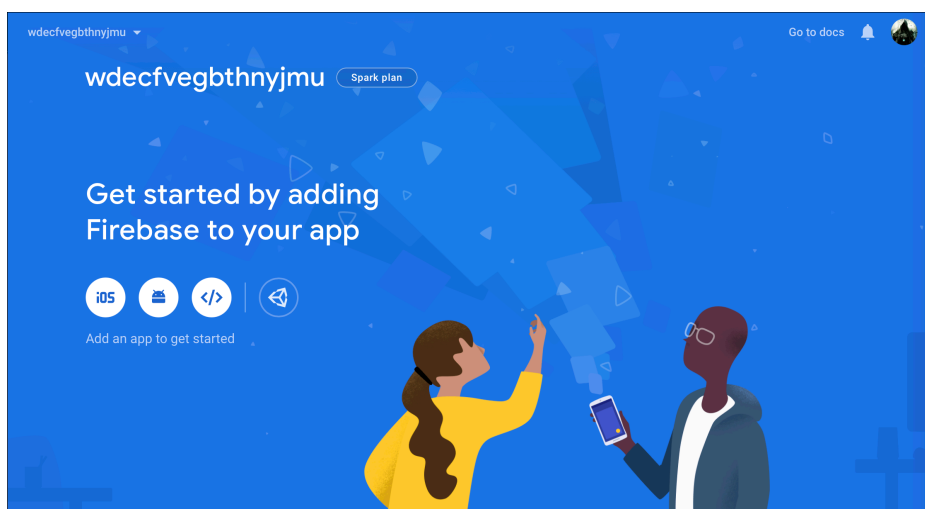


Рисунок 4.24 - створений проект

У новому вікні потрібно дати назву веб-додатку та після цього з'являться конфігурації(рис. 4.25) які потрібно скопіювати.

```
<!-- The core Firebase JS SDK is always required and must be listed
<script src="https://www.gstatic.com/firebasejs/8.6.5/firebase-app.js">

<!-- TODO: Add SDKs for Firebase products that you want to use
      https://firebase.google.com/docs/web/setup#available-libraries

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyAR4BY-X2EM1kA4KPSY9VlCqp3m1tsD_NA",
    authDomain: "wdecfvrgbthnyjmu.firebaseio.com",
    projectId: "wdecfvrgbthnyjmu",
    storageBucket: "wdecfvrgbthnyjmu.appspot.com",
    messagingSenderId: "136661993944",
    appId: "1:136661993944:web:c3e7f5db8e84ea874dcaad"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

Рисунок 4.25 - конфігурації

Тепер потрібно поєднати Firebase та Vue для цього після встановлення Vue потрібно відкрити файл *main.js* та вставити скопійовані конфігурації(рис. 4.26).

```
1  import { createApp } from 'vue'
2  import App from './App.vue'
3  import firebase from 'firebase'
4
5  // Your web app's Firebase configuration
6  const firebaseConfig = {
7    apiKey: "AIzaSyBwStTDTHPxb09CY-zqRlsnCqB4FK4UQo",
8    authDomain: "todolist-48177.firebaseio.com",
9    projectId: "todolist-48177",
10   storageBucket: "todolist-48177.appspot.com",
11   messagingSenderId: "1043718060190",
12   appId: "1:1043718060190:web:95f7effd2405d15dd8e38f"
13 };
14 // Initialize Firebase
15 firebase.initializeApp(firebaseConfig);
16
17 createApp(App).mount('#app')
18
```

Рисунок 4.26 - файл *main.js*

Також потрібно встановити firebase командою *npm install firebase*. Після цього, встановлення firebase завершено, Залишається тільки писати програмну реалізацію.

4.8 Результат роботи додатку

Інтерфейс централізованого веб-додатку виглядає наступним чином(рис. 4.27). Інтерфейс децентралізованого(рис. 4.28) веб-додатку так-само виглядає, але з права вверху є адрес акаунта який використовуємо для оплати транзакції.

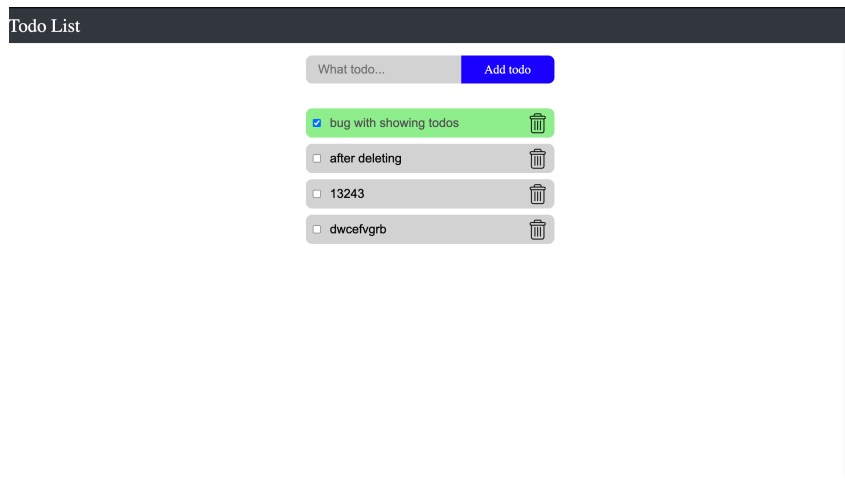


Рисунок 4.27 - централізований веб-додаток

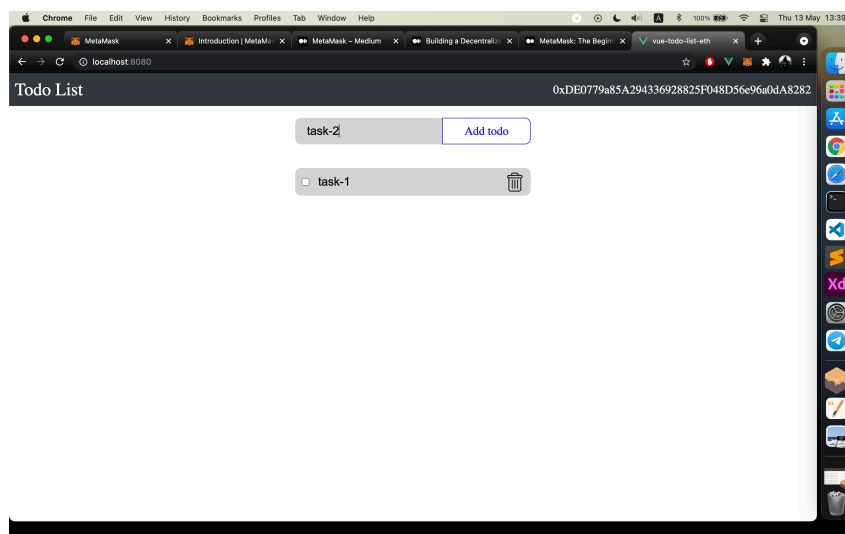


Рисунок 4.28 - децентралізований веб-додаток

Веб-додатки працюють правильно тільки є одна відмінність, у децентралізованому додатку за кожну дію створюється транзакція яку потрібно підтверджувати та за яку береться плата яку покриває акаунт в Ganache.

4.8.1 Створення нового завдання

В обох веб-додатках створимо нове завдання з назвою *new-task*. D

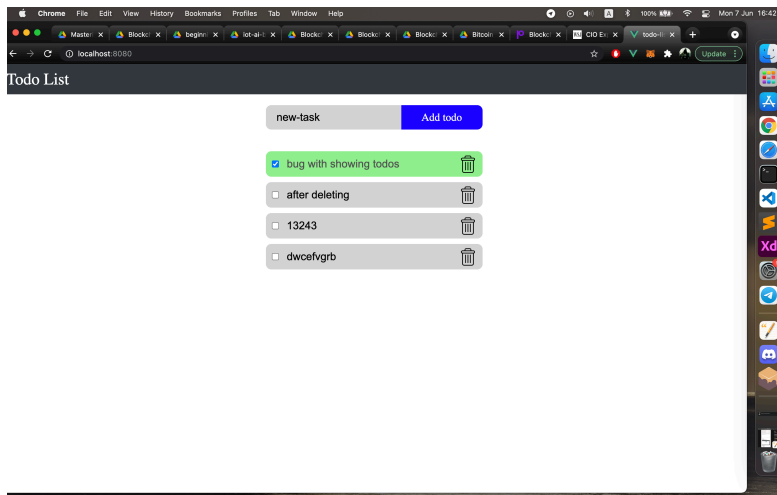


Рисунок 4.29 - створення нового завдання

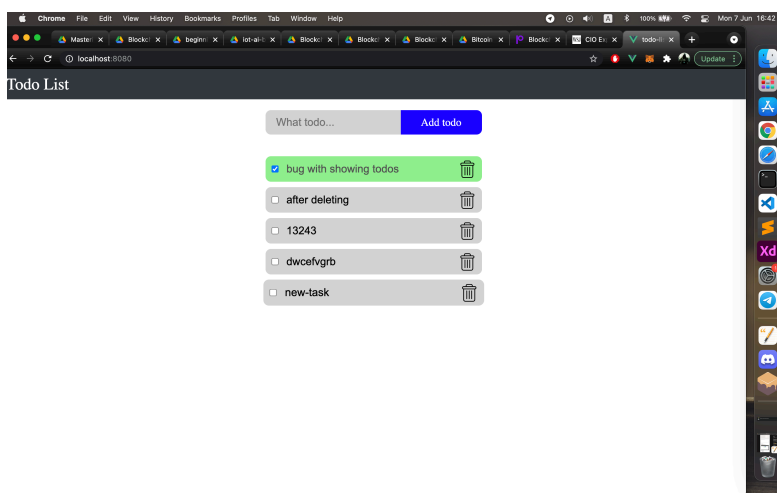


Рисунок 4.30 - нове завдання створено

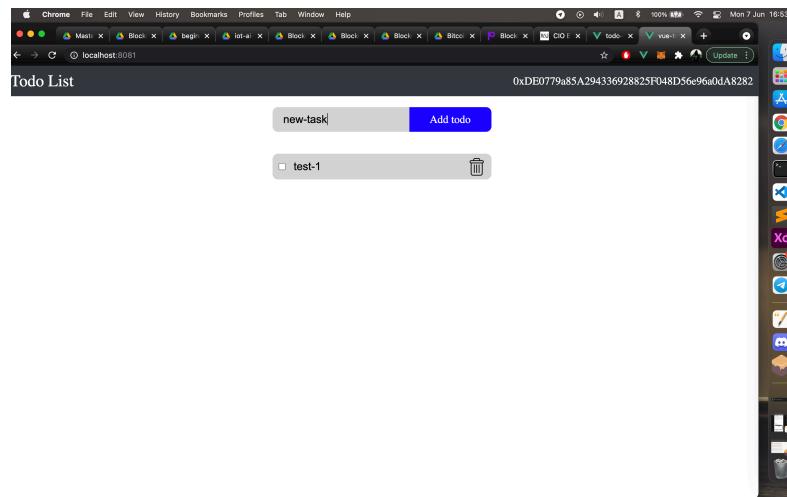


Рисунок 4.31 - створення нового завдання

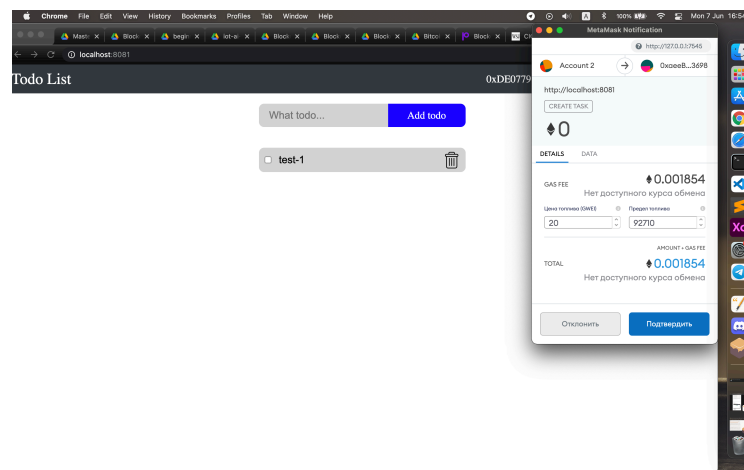


Рисунок 4.32 - підтвердження транзакції

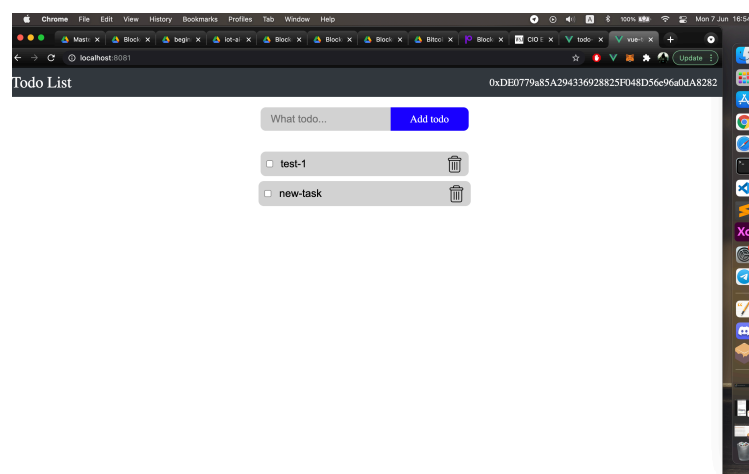


Рисунок 4.33 - нове завдання створено

Рисунки 4.29 та 4.30 це результат роботи централізованого веб-додатку, а рисунки з 4.31 до 4.33 децентралізованого. Повний огляд роботи веб-додатків знаходиться в ДОДАТОК А(для децентралізованого) та ДОДАТОК Б(для централізованого).

4.9 Висновки до розділу

У даному розділі було наведено реалізацію веб-додатка Todo-List двома способами. У централізованого веб-додатка стек технологій: Vue, Firebase. У децентралізованого веб-додатка стек технологій: Vue, Metamask, Solidity, Ganache, Truffle. Результат роботи в них практично однаковий, тільки у децентралізованого на кожну дію користувача потрібно підтверджувати транзакцію, яка має комісію.

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1 Постановка задачі проектування

Потрібно реалізувати додаток Todo-List. Програмний продукт призначений для запису модифікування та видалення завдань користувача. За допомогою алгоритма ФВА дізнаємось як краще це зробити.

5.2 Обґрунтування функцій та параметрів програмного продукту

Виходячи з мети, можна виділити основні функції програмного продукту:

- F1) вибір фронтенд-фреймворку
- F2) вибір бази даних
- F3) вибір середовища розробки

Кожна з функцій має два варіанти:

F1):

- а) Vue
- б) React

F2):

- а) Firebase
- б) MongoDB

F3):

- а) Visual Studio Code
- б) Sublime Text

Маючи такі функції та їх варіанти можна побудувати морфологічну карту(рис. 5.1), оскільки вона відображає всі можливі варіанти функцій.

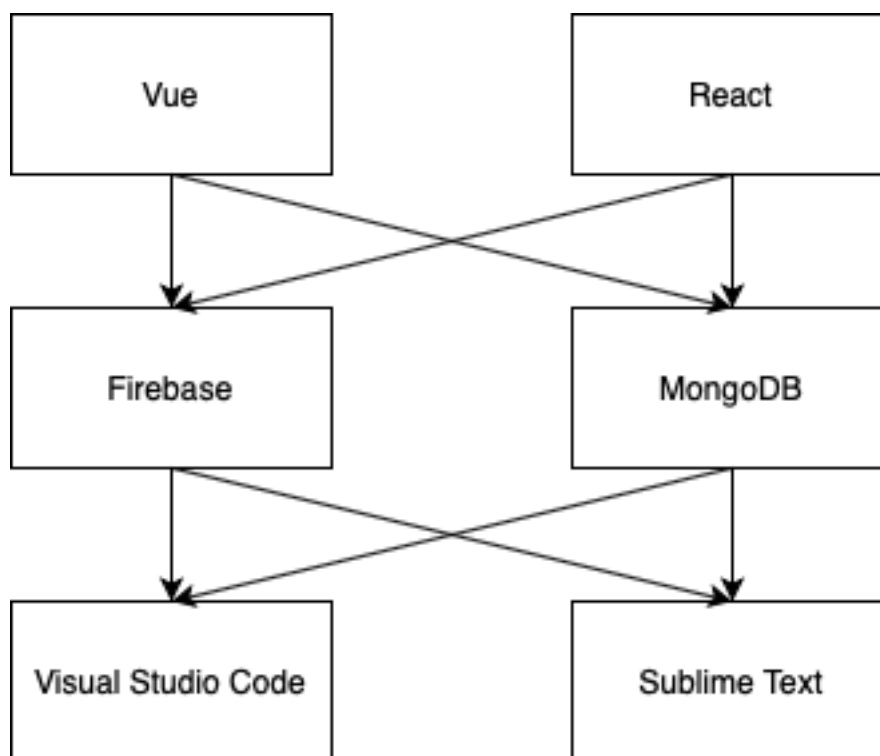


Рисунок 5.1 - морфологічна карта

Таблиця 5.1 - позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Фреймворк зовсім небагато важить, тому економиться час завантаження сторінки	Компонентний підхід не так гнучкий і очевидний, як в React.
	Б	Використовує віртуальне DOM дерево	JSX синтаксис
F2	А	Широкий спектр послуг та функцій	Обмежена передача даних
	Б	Простота середовища та швидке налаштування	Обмежений обсяг даних та розміщення
F3	А	редактор підтримує майже всі мови, що використовуються для створення додатків	Велике споживання оперативної пам'яті

Функції	Варіанти реалізації	Переваги	Недоліки
	Б	Небагато важить та легкий у використанні	Відсутність терміналу для компіляції коду

Маючи морфологічну карту побудуємо позитивно-негативну матрицю(Таб. 5.1). Дивлячись на таблицю деякі варіанти можна не враховувати.

Для функції F1 перевагу віддамо варіанту А, оскільки чим більше швидкість завантаження веб-сторінки тим краще для користувача.

Для функції F2 можна використати два варіанта.

Для функції F3 перевагу віддамо варіанту А, оскільки в варіанті Б немає вбудованого терміналу, доведеться паралельно відкривати термінал операційної система прописувати шлях до файлу потім його компілювати.

Після порівнянь залишились тільки два варіанти:

- F1 А , F2 А, F3 А
- F1 А , F2 Б, F3 А

5.3 Обґрунтування системи параметрів програмного продукту

Для характеристики програмного продукту використаємо параметри:

- X1 час компіляції
- X2 об'єм оперативної пам'яті
- X3 час написання коду
- X4 час виконання запитів у базу даних

Таблиця 5.2 - основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметру		
			Гірше	Середнє	Краще

Час компіляції	X1	мс	4500	3100	2000
Об'єм оперативної пам'яті	X2	Мб	80	55	35
Час написання коду	X3	Год	12	6	4
Час виконання запитів у базу даних	X4	мс	1200	520	210

Базуючись на таблиці(Табл. 5.2) побудуємо графіки, рис. 5.2 - рис. 5.5.

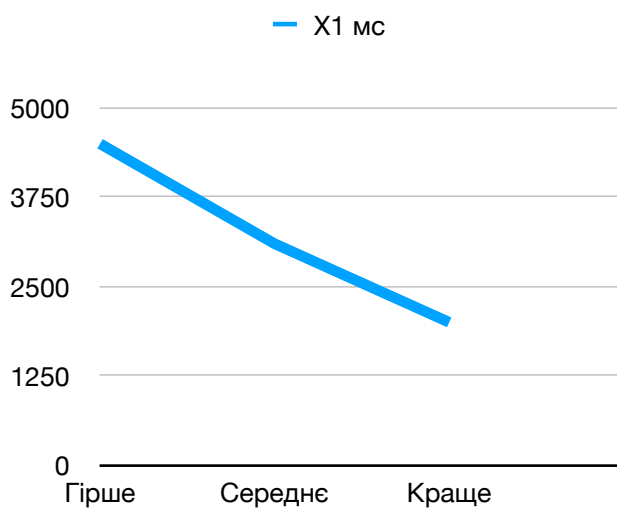


Рисунок 5.2 - час компіляції

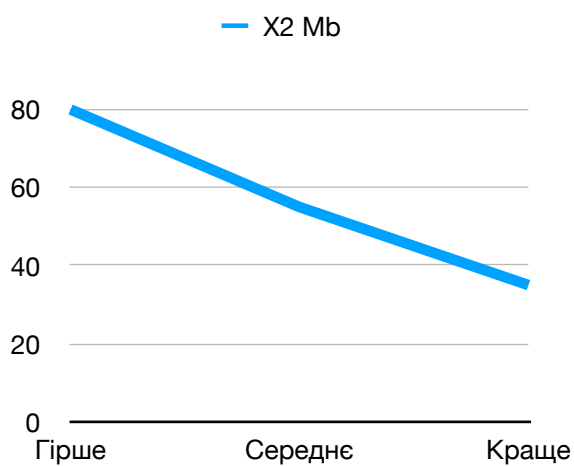


Рисунок 5.3 - об'єм оперативної пам'яті

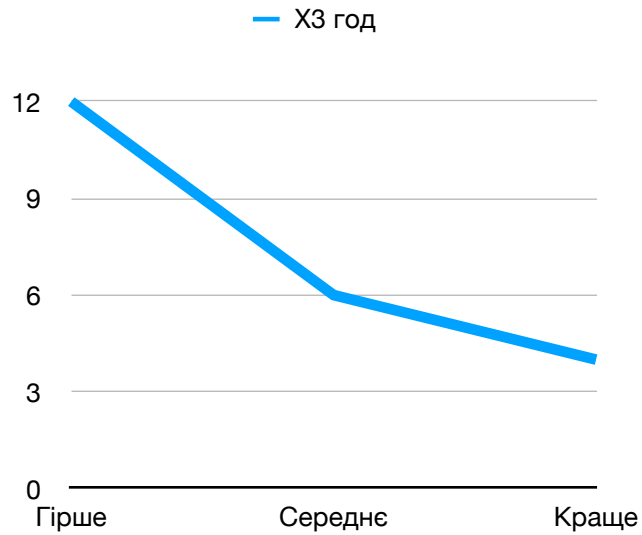


Рисунок 5.4 - час написання коду

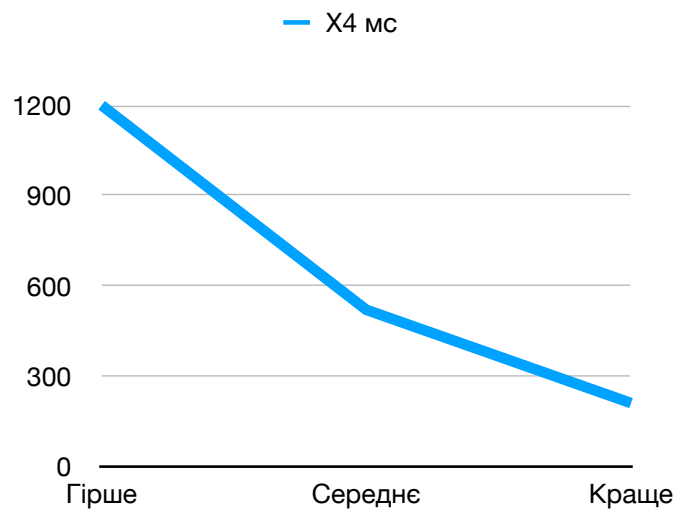


Рисунок 5.5 - Час виконання запитів у базу даних

5.4 Аналіз експертного оцінювання параметрів

За допомогою методу попарного порівняння визначимо значимість кожного параметра, де 4 – означає, що параметр має найбільш важливе значення, а 1 – найменш важливе.

Таблиця 5.3 - результат оцінки параметрів

Параметр	Ранг параметра по оцінці експерта							Сума рангів, Ri	Відхилення^	Квадрат відхилення
	1	2	3	4	5	6	7			
X1	1	1	2	2	1	2	1	10	-7.5	56.25
X2	2	3	1	1	2	1	2	12	-5.5	30.25
X3	3	2	4	4	3	3	3	22	4.5	20.25
X4	4	4	3	3	4	4	4	26	8.5	72.25
Разом	10	10	10	10	10	10	10	70	0	179

Порахуємо коефіцієнт узгодження:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0,73 > W_k = 0,67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт перевищує нормативний. Далі проведемо попарне порівняння всіх параметрів(Табл. 5.4) та розрахунок вагомості параметрів(Табл. 5.5).

Таблиця 5.4 - попарне зрівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове відхилення
	1	2	3	4	5	6	7		
X1 і X2	<	<	>	>	<	>	<	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	<	<	<	<	<	<	<	<	0.5
X2 і X3	<	>	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	<	<	>	>	<	<	<	<	0.5

Таблиця 5.5 - Розрахунок вагомості параметрів

Параметри X _i	Параметри X _j				Перша ітерація		Друга ітерація	
	1	2	3	4	В _i	К _{вi}	В _i	К _{вi}
X ₁	1	0.5	0.5	0.5	2	0.13	8.77	0.16
X ₂	1.5	1	0.5	0.5	3.5	0.22	11.5	0.2
X ₃	1.5	1.5	1	0.5	4.5	0.29	15.5	0.28
X ₄	1.5	1.5	1.5	1	5.5	0.35	20.5	0.36
Всього					15.5	1	56.27	1

Таблиця 5.6 - Розрахунок показників рівня якості заданих варіантів

Основні функції	Варіант реалізації функції	Параметр	Абсолютне значення параметра	Банальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт Рівня якості
F ₁	A	X ₁	9	5.6	0.15	0.65
F ₂	A	X ₂	15	9.6	0.21	3.5
	Б	X ₃	8	7.5	0.21	2.7
F ₃	A	X ₄	50	4.1	0.3	1.3

За даними з таблиці(Табл. 5.6), визначимо рівень якості варіантів:

$$K_{K1} = 0,65 + 3,5 + 1,3 = 5,45$$

$$K_{K2} = 0,65 + 2,7 + 1,3 = 4,65$$

З розрахунків, кращим є перший варіант, оскільки для нього коефіцієнт технічного рівня має найбільше значення.

5.5 Економічний аналіз розробки програмного забезпечення

Для визначення вартості розробки проведемо розрахунок трудомісткості:

Обидва варіанти включають в себе:

- 1) Розробка блокчейну для Todo-list
- 2) Написання коду відповідно з попереднім пункту
- 3) Тестування додатку

Завдання 1 відповідає групі А відповідно до ступеня новизни, та до 1 групи за складністю. Завдання 2 відноситься до групи В для 1 варіанту та групи А для другого, за складністю до групи 1. 3 завдання відповідає групі В, та складності 3. Для завдання 1 маємо: $T_p = 90$ людино-днів, $K_n = 1.7$, $K_{ск} = 1$, $K_{см} = 0,8$ Отримаємо:

$$T_1 = 90 * 1,7 * 1 * 0,8 = 122,4 \text{ людино-днів}$$

Провторимо для завдань 2 і 3. Для завдання 2 маємо: $T_p = 43$ людино-днів, $K_n = 0,81$, $K_{ск} = 1$, $K_{см} = 0,6$. Отримаємо:

$$T_2 = 43 * 0,81 * 1 * 0,6 = 20,9 \text{ людино-днів для 1 варіанту}$$

$$T_2 = 90 * 1,7 * 1 * 0,8 = 122,4 \text{ для 2 варіанту}$$

Для завдання 3 маємо: $T_p = 12$ людино-днів, $K_n = 0,5$, $K_{ск} = 1$, $K_{см} = 0,6$

Отримаємо:

$$T_3 = 12 * 0,5 * 1 * 0,6 = 3,6 \text{ людино-днів}$$

Складемо трудомісткість завдань, для кожного з варіантів

$$T_I = (122,4 + 20,9 + 3,6) * 8 = 1175,2 \text{ людино-годин}$$

$$T_{II} = (122,4 + 122,4 + 3,6) * 8 = 1987,2 \text{ людино-годин}$$

Варіант II має більшу трудомісткість. У розробці бере участь програміст з окладом 15200 грн. і експерт з тестування з окладом 12200 грн. Визначимо зарплату за годину:

$$CЧ = \frac{15200 + 12200}{2 \cdot 21 \cdot 8} = 81,54 \text{ грн.}$$

Розрахуємо заробітну плату розробників за варіантами:

$$\text{I. } C_{zn} = 1175,2 * 81,54 * 1,2 = 114\,990,97 \text{ грн.}$$

$$\text{II. } C_{zn} = 1987,2 * 81,54 * 1,2 = 194\,443,5 \text{ грн}$$

Відрахування на єдиний соціальний внесок(22%):

$$\text{I. } C_{vid} = 114\,990,97 * 0,22 = 25\,298,13 \text{ грн}$$

$$\text{II. } C_{vid} = 194\,443,5 * 0,22 = 42\,777,58 \text{ грн}$$

Визначимо витрати на оплату однієї машино-години (C_m). Оскільки одна машина обслуговує одного програміста з окладом 12500 грн. і коефіцієнтом зайнятості 0,2, отримаємо:

$$C_2 = 12 * 12500 * 0,2 = 30\,000 \text{ грн}$$

Порахуємо додаткову заробітну плату:

$$C_{zn} = 30\,000 * (1 + 0,2) = 36\,000 \text{ грн}$$

Відрахування на соціальний внесок

$$C_{vid} = 36\,000 * 0,22 = 7920 \text{ грн}$$

Амортизаційні відрахування отримуємо при амортизації 25% і вартості машини – 13000 грн.

$$C_a = 1,15 * 0,25 * 13\,000 = 3\,737,5 \text{ грн}$$

Витрати на ремонт:

$$C_p = 1,15 * 0,05 * 13000 = 747,5 \text{ грн}$$

Ефективний годинниковий фонд часу ПК за рік:

$$T_{ef} = (365 - 142 - 16) * 8 * 0,9 = 1677,6 \text{ год}$$

Витрати на оплату електроенергії:

$$C_{el} = 1677,6 * 0,2 * 0,61 * 3,52 = 561,93 \text{ грн}$$

Накладні витрати:

$$C_n = 13\,000 * 0,67 = 8710 \text{ грн}$$

Річні експлуатаційні витрати:

$$C_{експ} = 36\,000 + 7920 + 3737,5 + 747,5 + 561,93 + 8710 = 57\,676,93 \text{ грн}$$

Таким чином собівартість однієї ЕОМ складає:

$$C_{м-г} = 57\,676,93 / 1677,6 = 34,38 \text{ грн}$$

Оскільки всі роботи по розробці програмного продукту ведуться на ПК, розрахуємо витрати в залежності від обраного варіанту:

$$\text{I. } C_m = 34,38 * 1175,2 = 40\,403,37 \text{ грн}$$

$$\text{II. } C_m = 34,38 * 1987,2 = 68\,319,94 \text{ грн}$$

Накладні витрати(67%):

$$\text{I. } C_n = 114\,990,97 * 0,67 = 77\,043,95 \text{ грн}$$

$$\text{II. } C_n = 194\,443,5 * 0,67 = 130\,277,15 \text{ грн}$$

Вартість розробки програмного продукту відповідно до варіанту:

$$\text{I. } C_{nn} = 114\,990,97 + 25\,298,13 + 40\,403,37 + 77\,043,95 = 257\,736,42 \text{ грн}$$

$$\text{II. } C_{nn} = 194\,443,5 + 42\,777,58 + 68\,319,94 + 130\,277,15 = 435\,818,17 \text{ грн}$$

5.6 Вибір кращого варіанту

$$K_{тер1} = \frac{5,45}{257\,736,42} = 2,115 \cdot 10^{-5}$$

$$K_{тер2} = \frac{4,65}{435\,818,17} = 1,067 \cdot 10^{-5}$$

Найефективнішим виявився варіант 1 з коефіцієнтом техніко-економічного рівня рівня

$$K_{тер1} = 2,115 \cdot 10^{-5}$$

Висновки до розділу

Отже після виконання функціонально-вартісного аналізу було виявлено, що перший варіант є оптимальним. Його показник $2,115 \cdot 10^{-5}$ виявився найбільшим. Даний варіант має такі параметри:

- вибір фреймворку Vue
- вибір firebase як бази даних
- вибір VS Code

ВИСНОВКИ ПО РОБОТІ

Під час виконання дипломної роботи, було досліджено блокчейн, його переваги та недоліки. З переваг можна відмітити безпеку даних та стійкість до атак. Безпека даних, тут після того як блок з даними був доданий до блокчейну його не можливо змінити чи видалити. Стійкість до атак, беручи до уваги централізовану систему, якщо був атакований чи пошкоджений головний сервер то система не буде працювати на відміну від децентралізованої, оскільки вона за рахунок своєї децентралізації немає головного серверу, тому, наприклад, якщо буде атакований один вузол то всі інші вузли будуть працювати, а значить система також. З недоліків можна відмітити масштабованість, оскільки децентралізована система це вузли які спілкуються між собою в одноранговій мережі і кожен із них зберігає копію даних, а вузли в свою чергу це комп'ютери в яких є жорсткі диски і звідси вони мають обмежену пам'ять. Оскільки копія даних може потребувати більше пам'яті ніж є на комп'ютері, тому із цього слідує недолік.

Також були розглянуті реальні кейси використання централізованих та децентралізованих систем і в кожній були вказані переваги децентралізованих систем. Особливо із них виділяється кейс про вибори президента.

У ході дослідження було проаналізовано інструментарій блокчейну для розробки веб-додатків. Було обрано один із простіших веб-додатків для ілюстрації технічних моментів використання блокчейну. Було реалізовано веб-додаток Todo-List двома способами: централізовано та децентралізовано. Веб-додаток може додавати, модифікувати, видаляти та змінювати статус виконання завдання.

ПЕРЕЛІК ПОСИЛАНЬ

1. CIO Explainer: What Is Blockchain? [Електроний ресурс] / Steven Norton // The Wall Street Journal - 2017. - Режим доступу до ресурсу: <https://www.wsj.com/articles/BL-CIOB-8993>.
2. Andreas M. Antonopoulos Mastering Bitcoin: Programming the Open Blockchain // O'Reilly Media - 2017. - №2. - С. 171.
3. Blockchain Tutorial: Learn Blockchain Technology (Examples) [Електроний ресурс] / Bob Peterson // Guru99 - 2018. - Режим доступу до ресурсу: <https://www.guru99.com/blockchain-tutorial.html>.
4. What Is Blockchain Technology? [Електроний ресурс] / Jack Davis // Cbinsights - 2020. - Режим доступу до ресурсу: <https://www.cbinsights.com/research/what-is-blockchain-technology/>.
5. Blockchain – Hot or Not? [Електроний ресурс] / Sergii Solomakha // Softengi- 2019. - Режим доступу до ресурсу: <https://softengi.com/blog/blockchain-hot-or-not/>.
6. Blockchain – Hot or Not? [Електроний ресурс] / Harry Moore // Binance Academy - 2020. - Режим доступу до ресурсу: <https://academy.binance.com/en/articles/history-of-blockchain>.
7. Nishith Pathak, Anurag Bhandari. IOT, AI, and blockchain for .NET // Apress - 2018. - №1. - С. 195-196.
8. Centralized vs Decentralized vs Distributed Systems [Електроний ресурс] / Manfred Touron // Berty - 2019. - Режим доступу до ресурсу: <https://berly.tech/blog/decentralized-distributed-centralized/>.
9. Centralized vs Decentralized: What's the Difference? [Електроний ресурс] / Laura M. // BitDegree - 2021. - Режим доступу до ресурсу: <https://www.bitdegree.org/crypto/tutorials/centralized-vs-decentralized>.
10. Как создать приложение с использованием блокчейна [Електроний ресурс] / Марк Денисенко // Umbrella IT - 2018. - Режим доступу до ресурсу: <https://umbrellait.com/ru/blog/how-to-build-a-blockchain-app/>.

11. List Of 10 Best Blockchain Tools [Электронный ресурс] / Aviv Lichtigstein // 101 Blockchains - 2020. - Режим доступа до ресурсу: <https://101blockchains.com/best-blockchain-tools/>.
12. How to Build Blockchain App - Ethereum Todo List 2019 [Электронный ресурс] / Gregory McCubbin // Dapp University - 2021. - Режим доступа до ресурсу: <https://www.dappuniversity.com/articles/blockchain-app-tutorial>.
13. Разработка децентрализованных приложений [Электронный ресурс] / JazzTeam // JazzTeam Software Development Company - 2018. - Режим доступа до ресурсу: <https://jazzteam.org/ru/technical-articles/development-of-decentralized-applications/#8>.
14. Vikram Dhillon, David Metcalf, Max Hooper. Blockchain Enabled Applications // Apress - 2017. - №1. - С. 36.
15. Build your Ethereum DApp on Windows with Truffle, Ganache and Metamask (Beginners Guide) [Электронный ресурс] / Filza Tariq // Medium - 2019. - Режим доступа до ресурсу: <https://medium.com/@filzatariq92>.
16. What are smart contracts on blockchain? [Электронный ресурс] / IBMteam // IBM- 2019. - Режим доступа до ресурсу: <https://www.ibm.com/topics/smart-contracts>.

ДОДАТОК А

На рисунках 1 - 3 продемонстровано процес створення нового завдання(task-2).

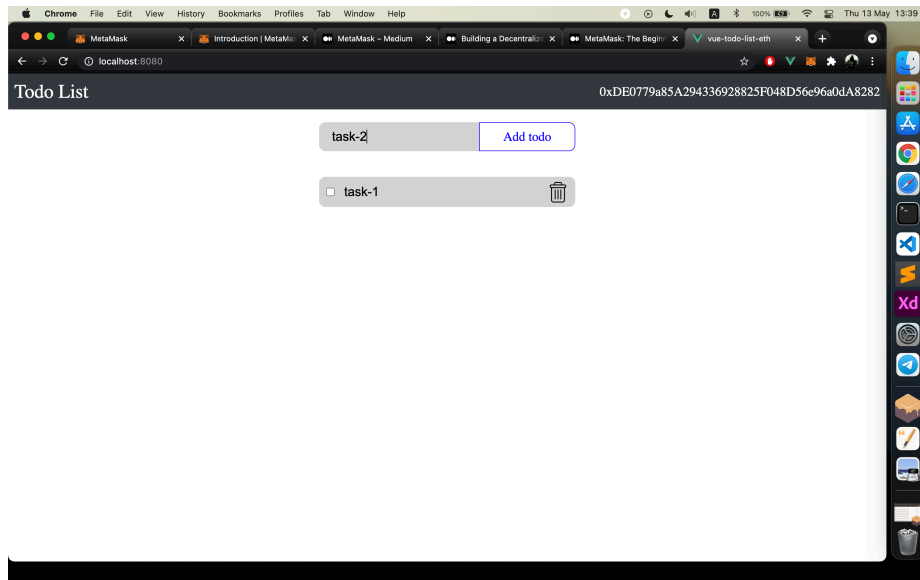


Рисунок 1 - Створення нового завдання

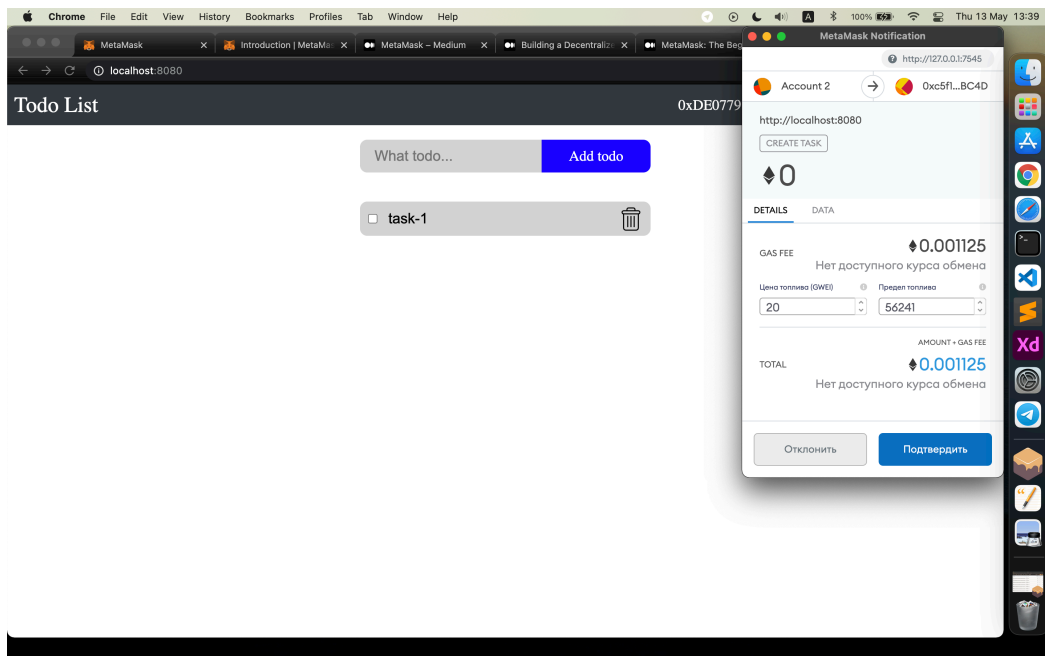


Рисунок 2 - Підтвердження транзакції

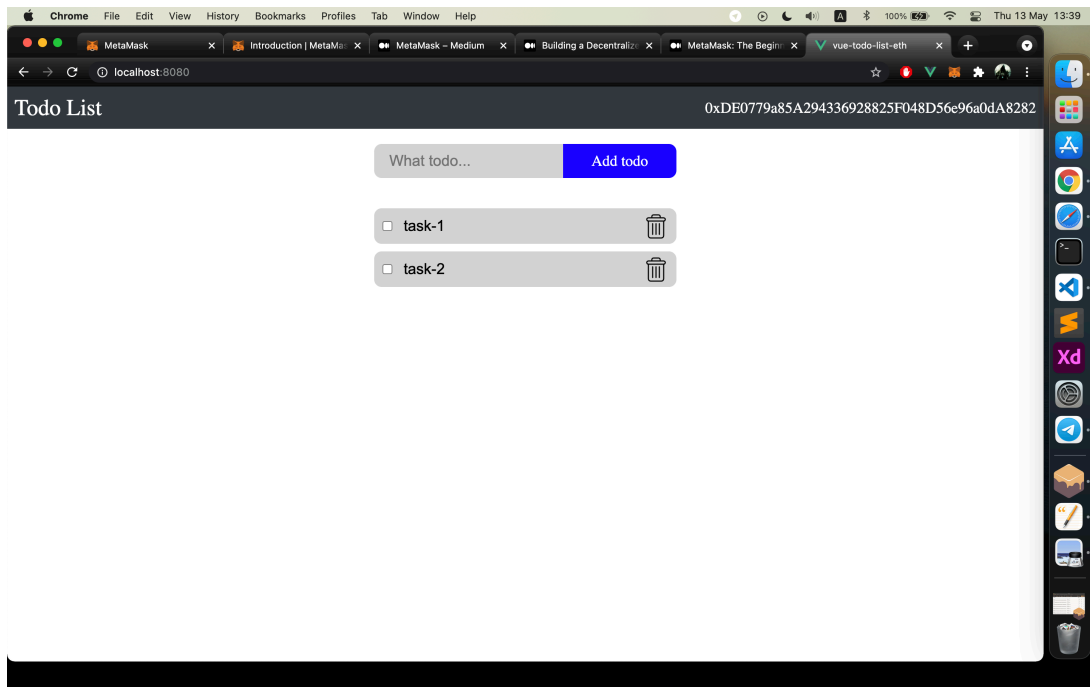


Рисунок 3 - Створено нове завдання

На рисунках 4 - 6 продемонстровано процес оновлення завдання(task-2) на завдання(task-2-updated).

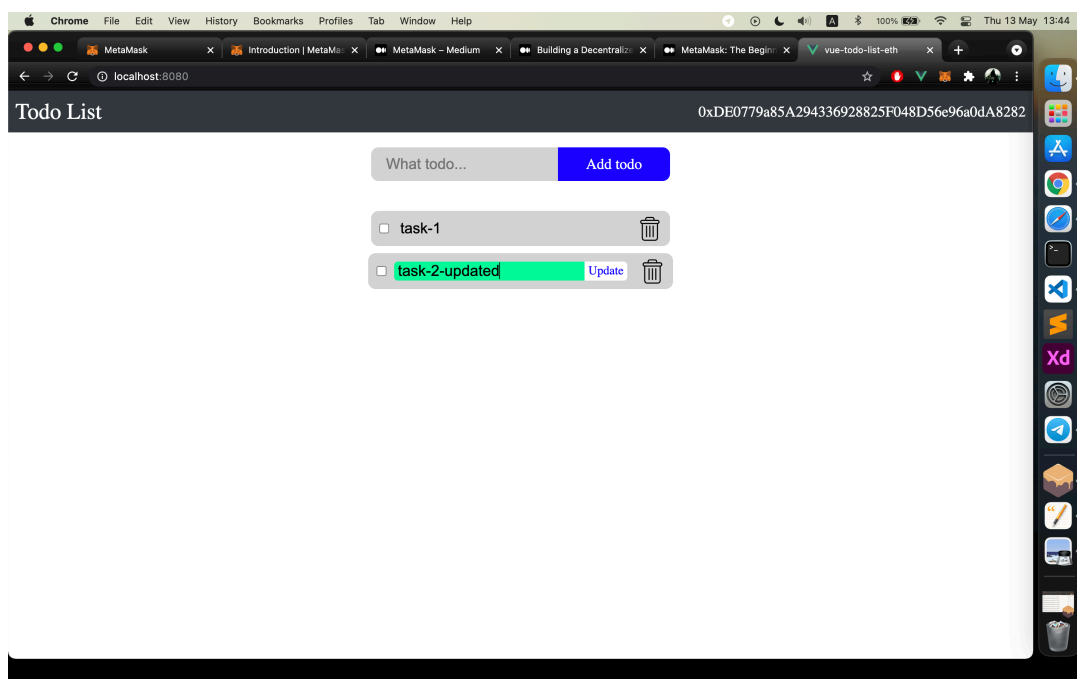


Рисунок 4 - оновлення завдання

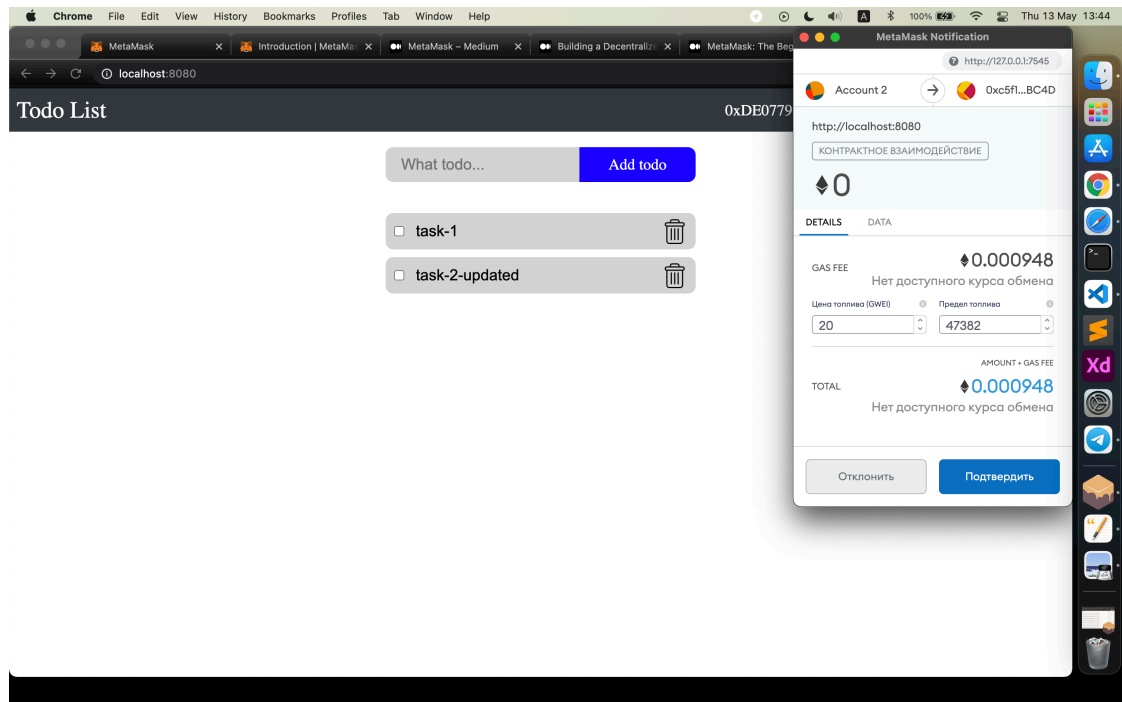


Рисунок 5 - підтвердження транзакції

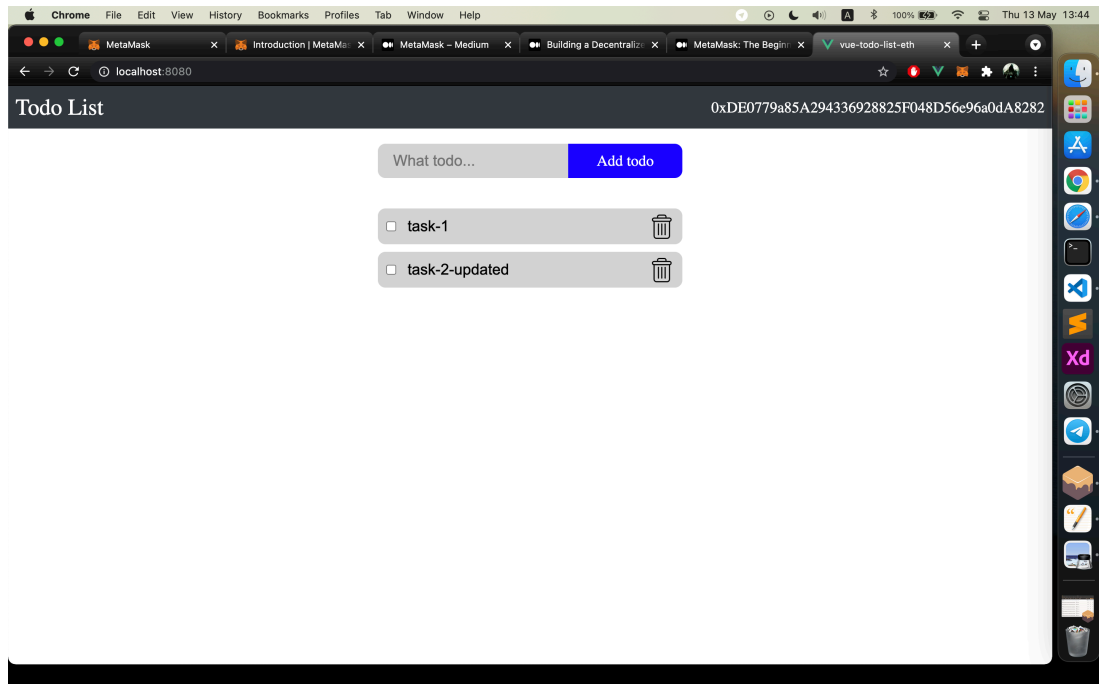


Рисунок 6 - завдання оновлено

На рисунках 7 - 9 продемонстровано зміну стану виконання завдання(task-2-updated).

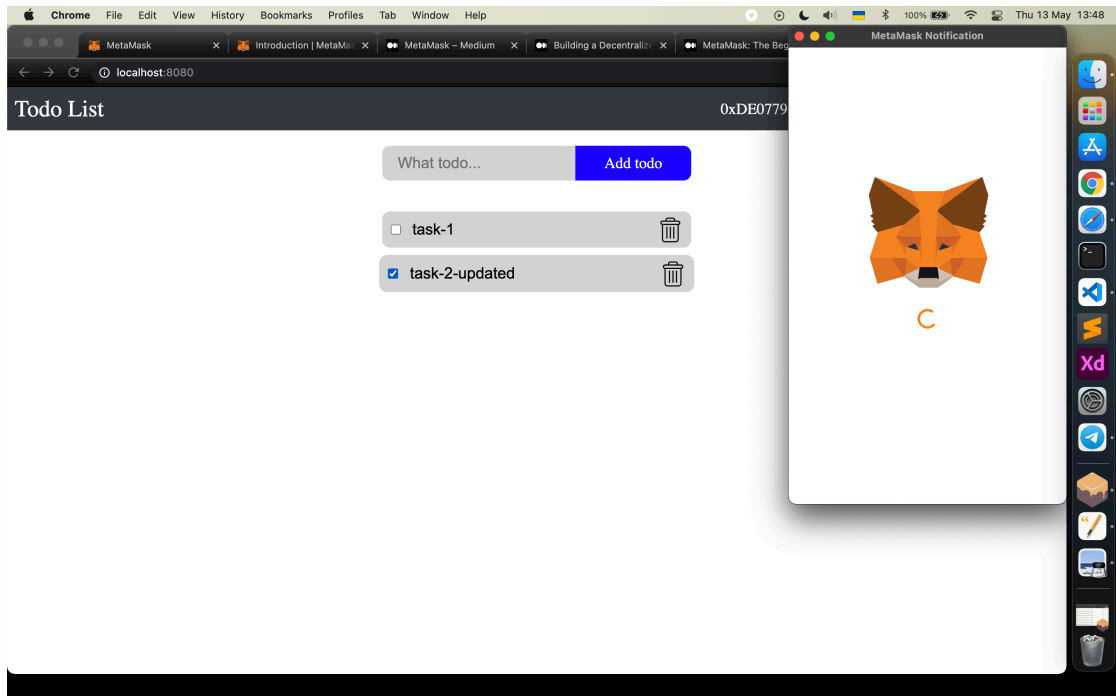


Рисунок 7 - зміна стану виконання завдання

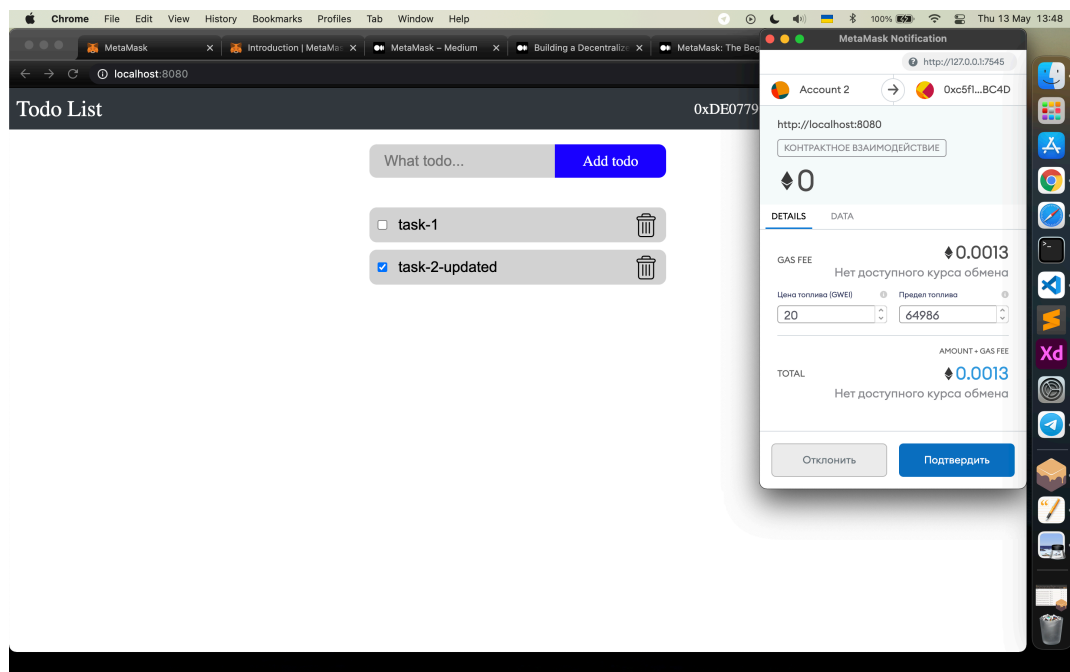


Рисунок 8 - підтвердження транзакції

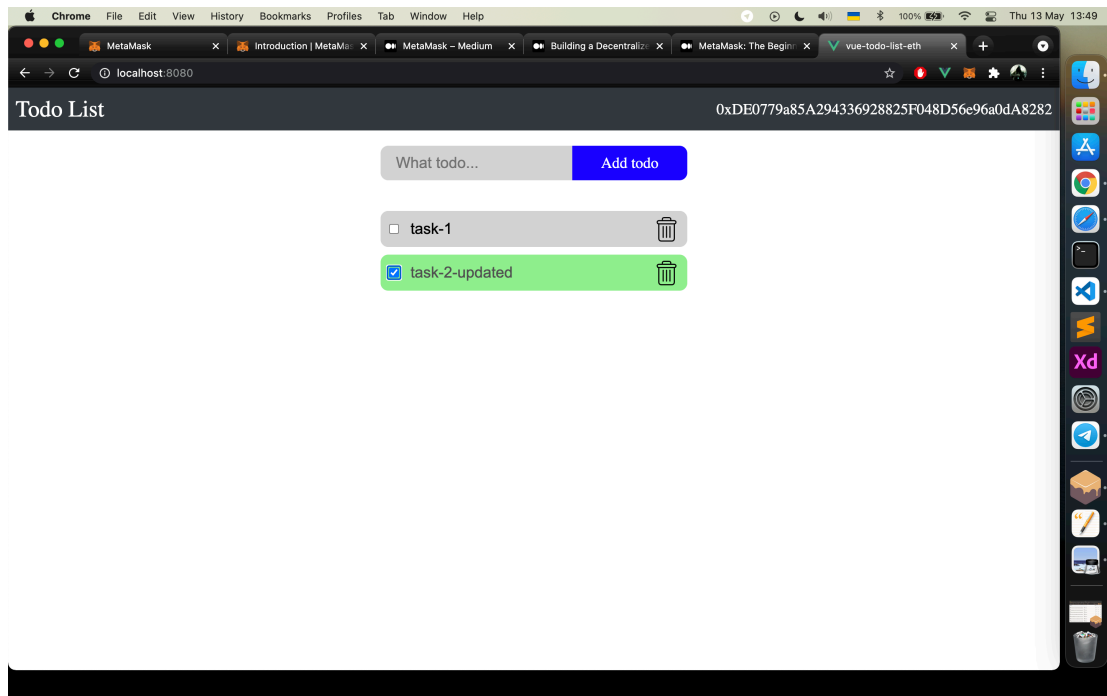


Рисунок 9 - змінений стан виконання завдання

На рисунках 10 - 12 продемонстровано видалення завдання(task-2-updated).

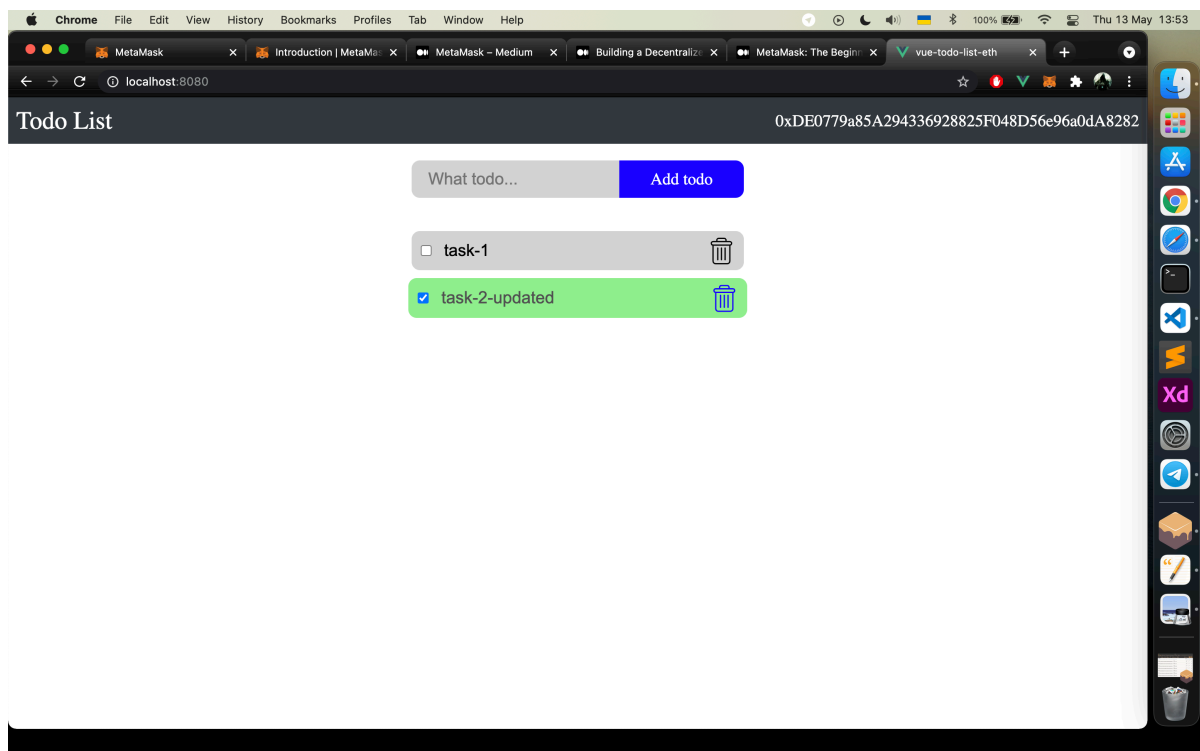


Рисунок 10 - видалення завдання

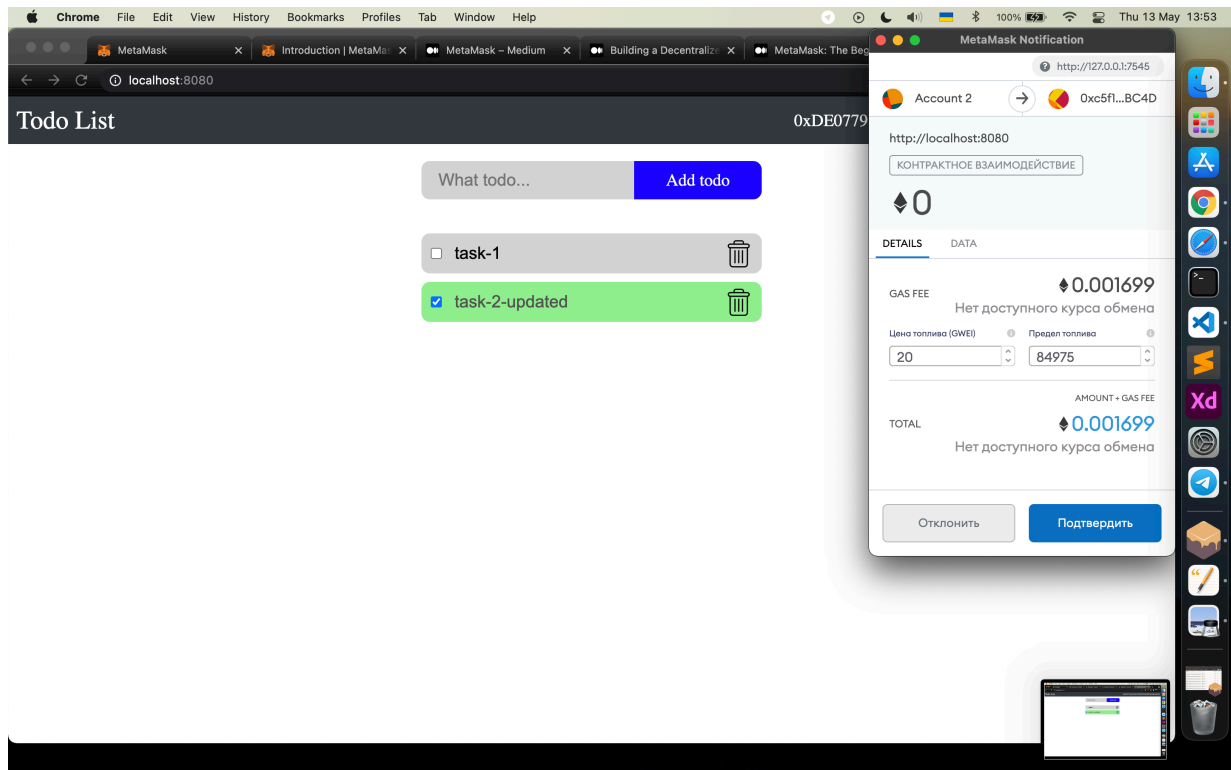


Рисунок 11 - підтвердження транзакції

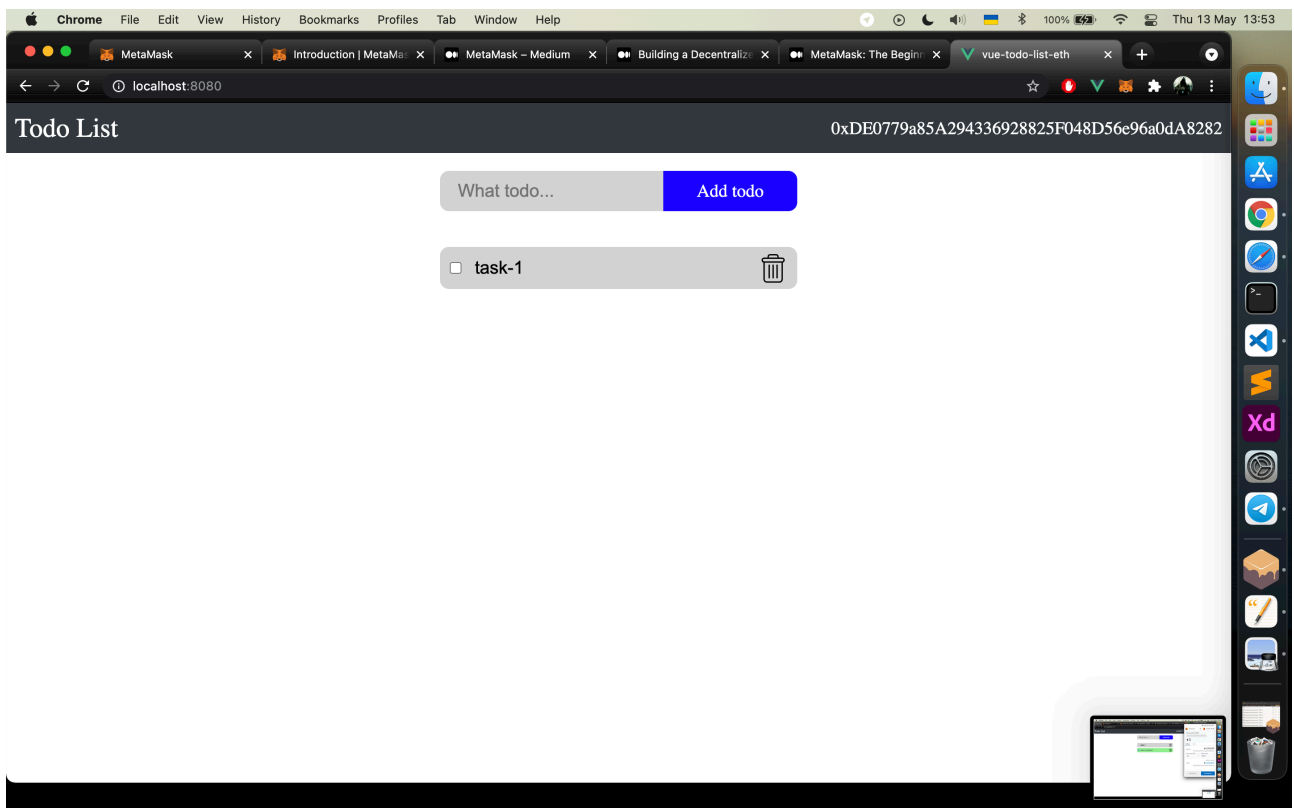


Рисунок 12 - завдання видалено

ДОДАТОК Б

На рисунках 1 - 2 продемонстровано процес створення нового завдання(task-2).

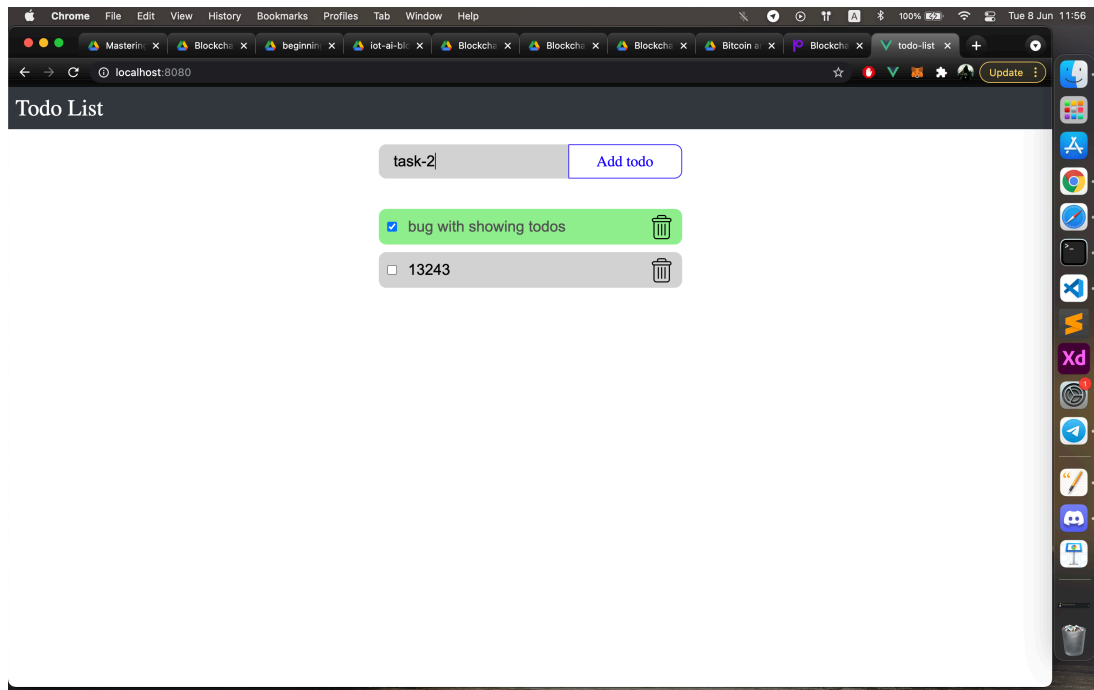


Рисунок 1 - Створення нового завдання

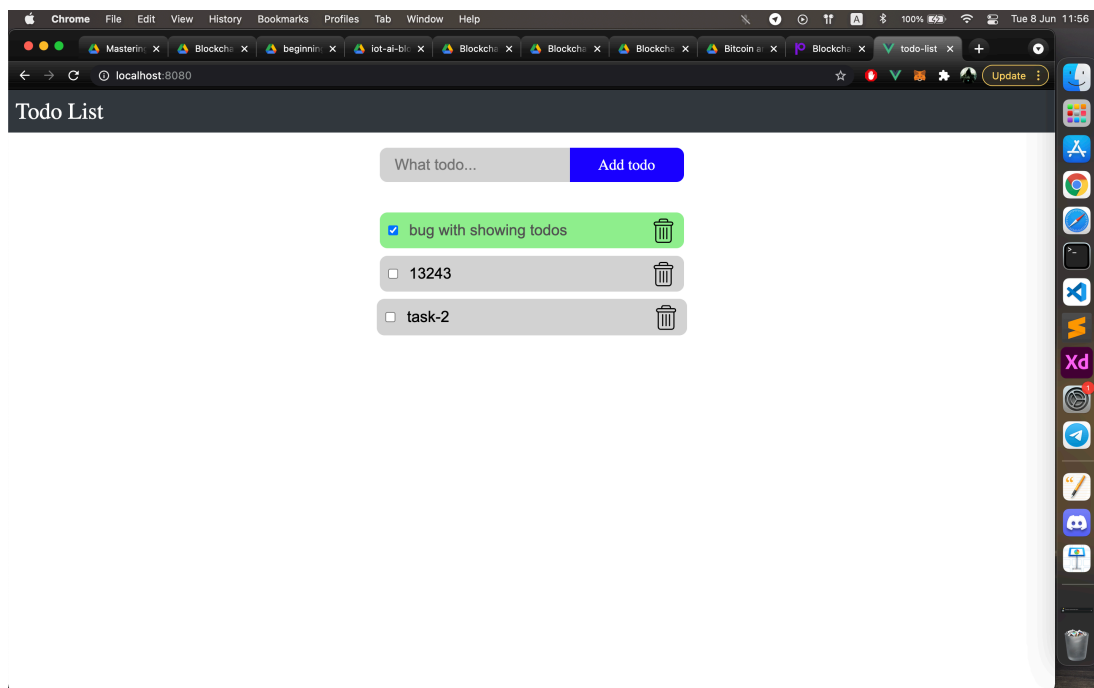


Рисунок 2 - Створено нове завдання

На рисунках 3 - 4 продемонстровано процес оновлення завдання(task-2) на завдання(task-2-updated).

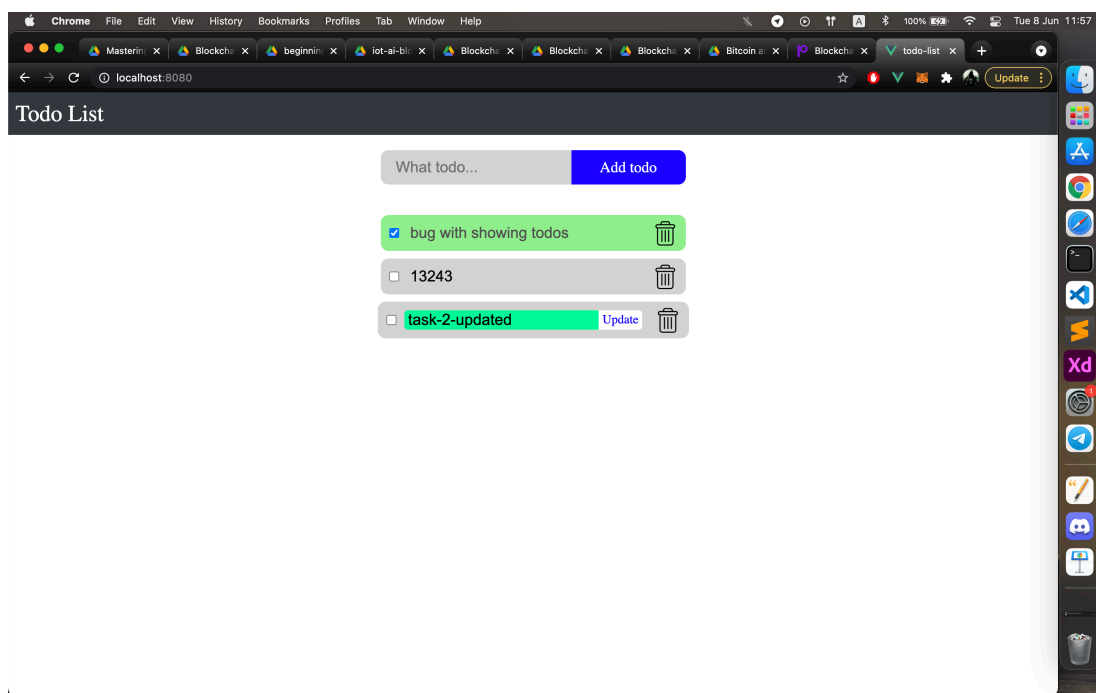


Рисунок 4 - оновлення завдання

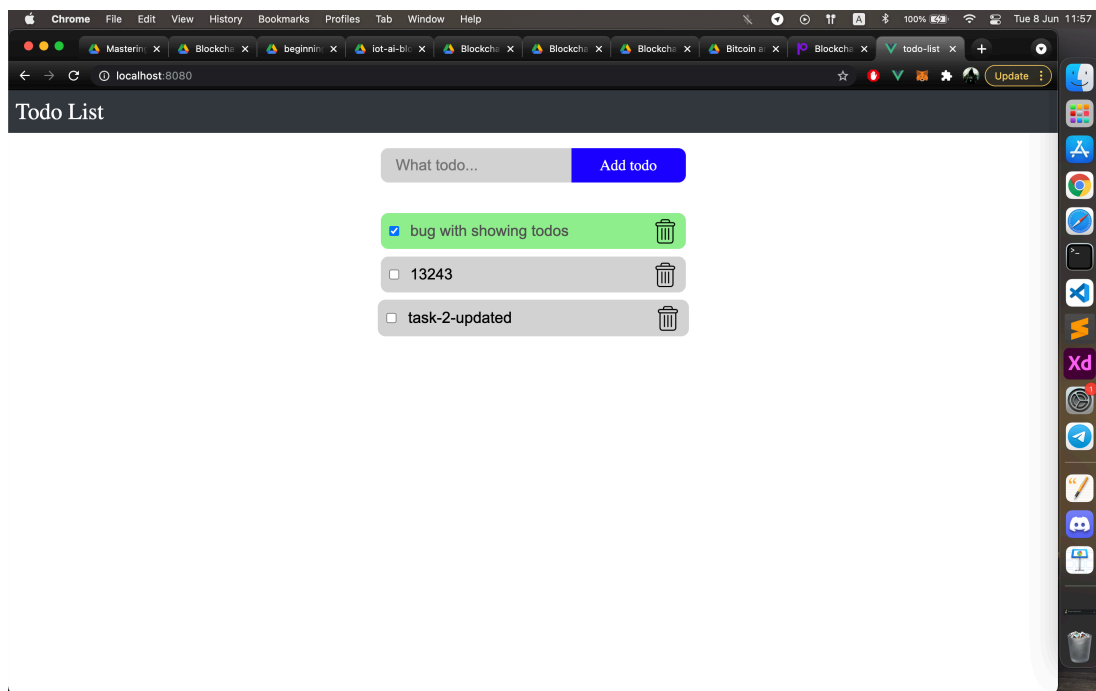


Рисунок 5 - завдання оновлено

На рисунках 6 - 7 продемонстровано зміну стану виконання завдання(task-2-updated).

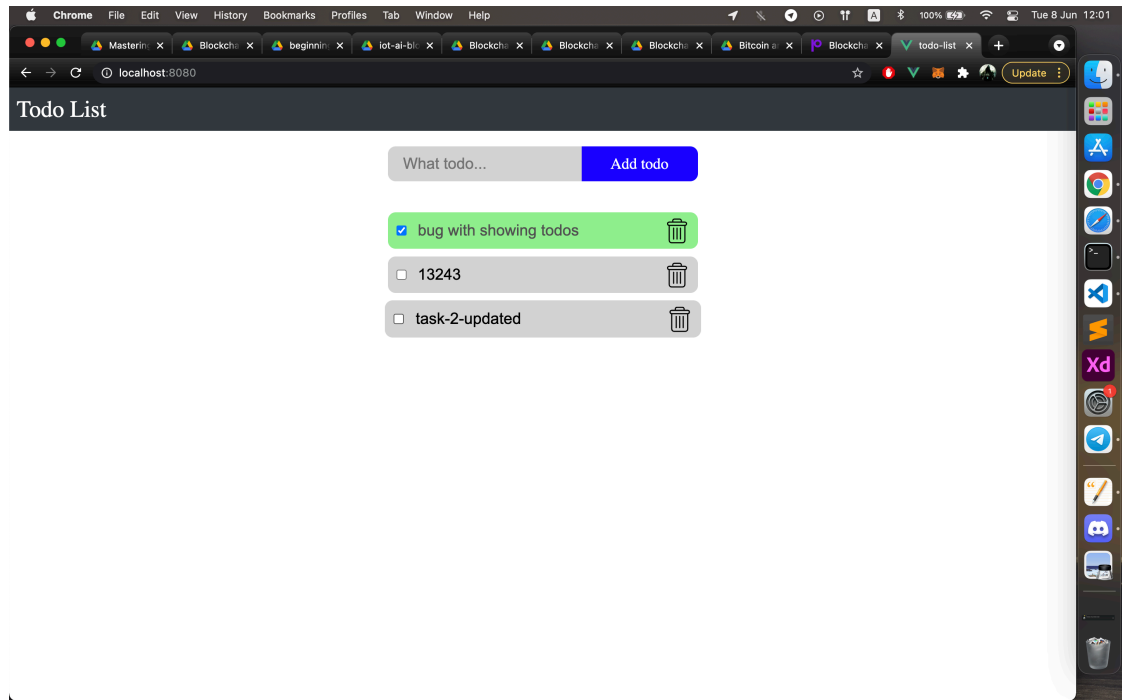


Рисунок 6 - зміна стану виконання завдання

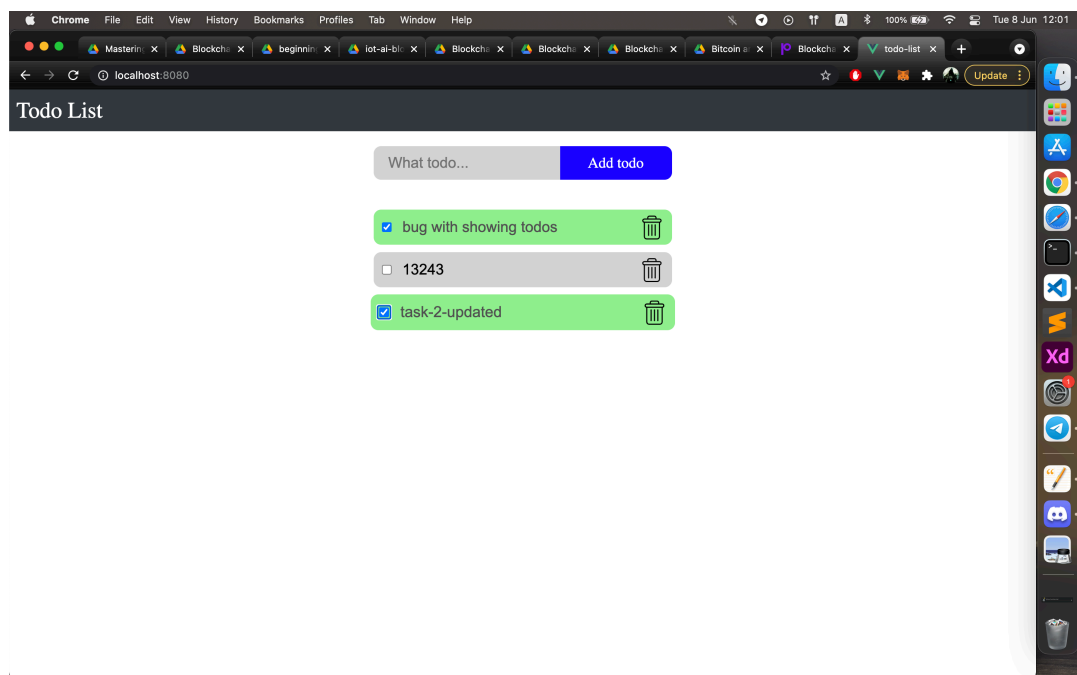


Рисунок 7 - змінений стан виконання завдання

На рисунках 8 - 9 продемонстровано видалення завдання(task-2-updated).

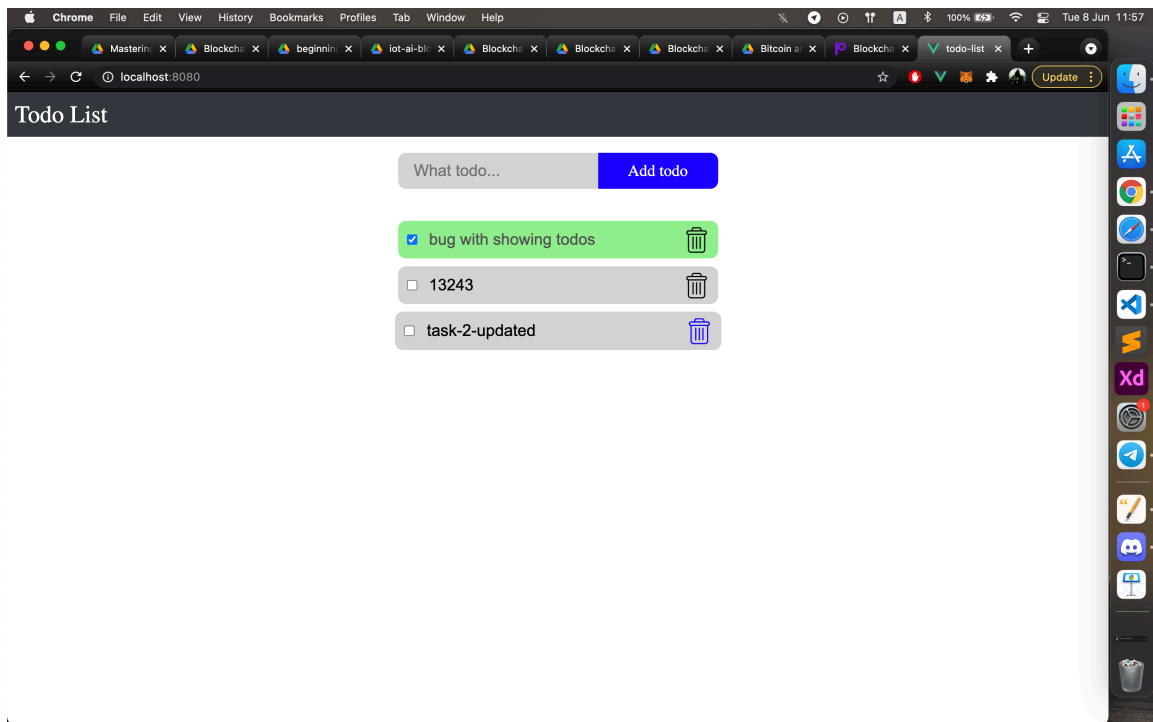


Рисунок 8 - видалення завдання

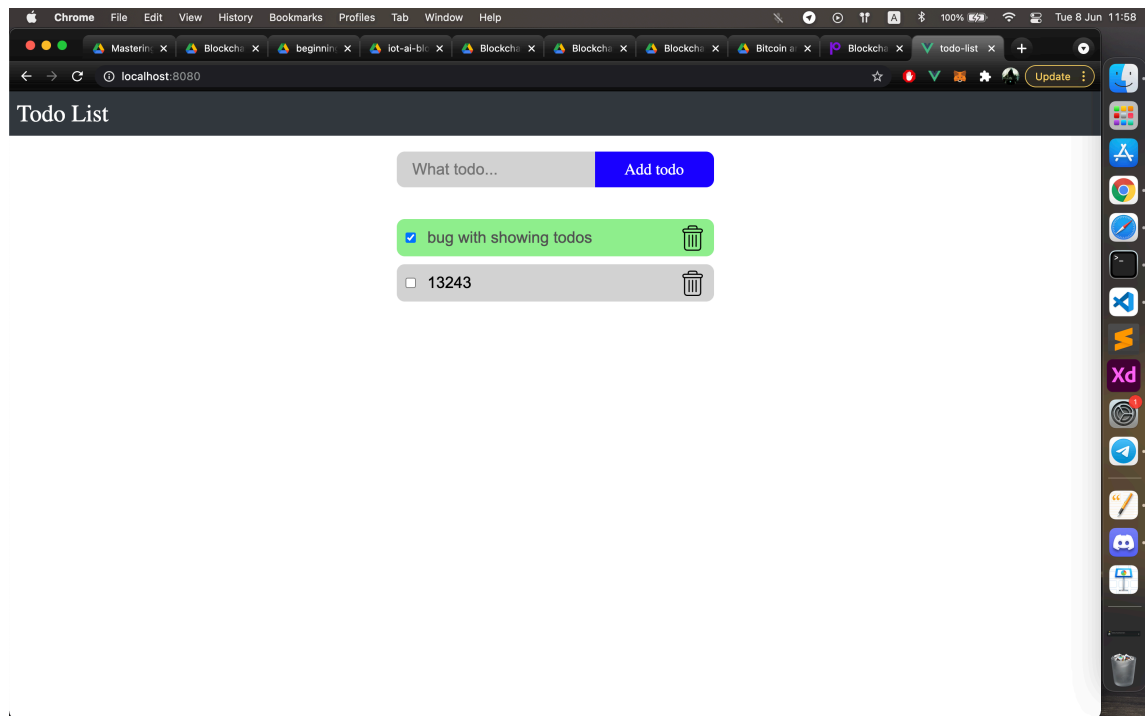


Рисунок 9 - завдання видалено