**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**
**АВТОМАТИЗОВАНИХ СИСТЕМ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження можливостей застосування децентралізованих технологій в розробці сучасного програмного забезпечення»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
*(код, найменування спеціальності)*
освітньо-професійної програми Інформаційні системи та технології
*(назва)*

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Ілля РОМАНЕНКО
*(підпис)* *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Ілля РОМАНЕНКО
група ІСДМ-64

Керівник: Ольга ПОЛОНЕВИЧ
*науковий ступінь, вчене звання* к.т.н., доцент

Рецензент:
*науковий ступінь, вчене звання* _____
Ім'я, ПРІЗВИЩЕ

**Київ 2023**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

**ЗАТВЕРДЖУЮ**
Завідувач кафедрою ІПЗАС
_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Романенку Іллі Миколайовичу
*(прізвище, ім'я, по батькові здобувача)*

1. Тема кваліфікаційної роботи: Дослідження можливостей застосування децентралізованих технологій в розробці сучасного програмного забезпечення

керівник кваліфікаційної роботи Ольга ПОЛОНЕВИЧ к.т.н., доцент,
*(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)*
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література,теорія проектування інформаційних систем

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   Аналіз особливостей децентралізованих мереж
   Дослідження технологій, програмного та апаратного забезпечення для розробки децентралізованих додатків
   Реалізація прототипу інформаційної системи
5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Підбір технічної літератури | 19.10.2023р. 05.11.2023р. | |
| 2. | Вивчення матеріалів для аналізу | 06.11.2023р. 14.11.2023р. | |
| 3. | Аналіз програмних, технічних рішень для децентралізованих додатків | 14.11.2023р. 21.11.2023р. | |
| 4. | Аналіз варіантів застосування | 22.11.2023р. 25.11.2023р. | |
| 5. | Вступ, висновки, реферат | 25.11.2023р. 01.12.2023р. | |
| 6. | Розробка прототипу | 01.12.2023р. 12.12.2023р | |
| 7. | Розробка демонстраційного матеріалу, доповідь | 12.12.2023р. 14.12.2023р. | |

Здобувач вищої освіти _____ Ілля РОМАНЕНКО
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 85 стор., 0 табл., 29 рис., 12 джерел.

*Мета роботи* – дослідження потенціалу децентралізованих технологій у розробці програмного забезпечення для інтернет-технологій.

*Об'єкт дослідження* – процес розробки програмного забезпечення.

*Предмет дослідження* – застосування децентралізованих технологій для веб-розробки.

*Короткий зміст роботи*: У роботі досліджено та оцінено децентралізовані технології, підходи до створення інформаційних систем, орієнтованих на веб. Здійснено аналіз ефективності застосування різноманітних технологій для створення функціональних та надійних веб-додатків, включно з оглядом переваг використання певного програмного забезпечення, фреймворків і інструментів, які спрощують процес розробки.

КЛЮЧОВІ СЛОВА: ДЕЦЕНТРАЛІЗОВАНІ ТЕХНОЛОГІЇ, БЛОКЧЕЙН РОЗПОДІЛЕНЕ ОБЧИСЛЕННЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІННОВАЦІЇ, БЕЗПЕКА, МАСШТАБОВАНІСТЬ.

## ABSTRACT

Text part of the master's qualification work: 85 pages, 29 pictures, 0 table, 12 sources.

*The purpose of the work* – exploring the potential of decentralized technologies in software development for internet technologies.

*Object of research* – software development process.

*Subject of research* – application of decentralized technologies for web development.

*Summary of the work*: In the work, decentralized technologies and approaches to creating web-oriented information systems were researched and evaluated. An analysis of the effectiveness of applying various technologies for creating functional and reliable web applications was carried out, including a review of the advantages of using certain software, frameworks, and tools that simplify the development process.



KEYWORDS: DECENTRALIZED TECHNOLOGIES, BLOCKCHAIN, DISTRIBUTED COMPUTING, SOFTWARE ENGINEERING, INNOVATIONS, SECURITY, SCALABILITY.

ЗМІСТ

# ВСТУП

*Актуальність роботи*. У цьому магістерській роботі розглядається сучасний період стрімких змін у сфері інформаційних технологій (ІТ), який характеризується революційними інноваціями та радикальною трансформацією традиційних підходів до розробки програмного забезпечення. Основну увагу в цьому контексті привертає впровадження та розвиток децентралізованих технологій, особливо блокчейну та розподілених обчислень, які відіграють ключову роль у сучасному ІТ-ландшафті.

Ці передові технології відкривають новітні можливості для підвищення ефективності, безпеки та прозорості програмних систем, відповідаючи таким чином на зростаючі вимоги до якості та надійності ІТ-продуктів. Децентралізація, як основа цих технологій, пропонує інноваційні підходи для вирішення таких важливих викликів, як масштабування, безпека та цілісність даних, що має фундаментальне значення для сучасної програмної інженерії.

У вступній частині цього магістерського проекту сформовано основні проблемні питання, визначаємо актуальність та значущість децентралізованих технологій у сучасному технологічному ландшафті, а також окреслюємо цілі та задачі дослідження. Ці задачі спрямовані на розробку та впровадження інноваційних рішень у сфері програмного забезпечення, що дозволить відповісти на сучасні виклики та потреби галузі.

*Завдання* даної магістерської роботи є:

- аналіз новаторських стратегій у розробці веб-орієнтованих систем;

- опис можливих способів використання;

- з`ясування можливих переваг та недоліків уже існуючих програмних засобів, середовищ та інструментів для розробки інформаційних систем із застосуванням децентралізованих технологій.

- розробка веб-додатку за допомогою сучасних технологічних інструментів.

*Мета роботи* – створення web-додатку для спілкування, використовуючи інструменти на основі Subsocial SDK.

*Об'єкт дослідження* - процес розробки web-додатку.

*Предмет дослідження* - інструментарій децентралізованих засобів розробки програмного забезпечення для створення веб-додатку.

*Методи дослідження.* Аналітичні дослідження.

*Наукова новизна* одержані результати відкривають шляхи застосування інноваційних методів у розробці програмного забезпечення в браузерному середовищі, використовуючи альтернативні стратегії для його створення.

*Практичне цінність*. Досягнення цих результатів включає створення веб-додатку для онлайн-комунікації, який буде доступний користувачам цієї системи. Розробники інформаційних систем можуть використовувати елементи цієї магістерської роботи як у процесі створення нових програмних продуктів, так і для повторного застосування розроблених програмних компонентів у майбутніх проектах.

# 1 ПРЕДМЕТНА ОБЛАСТЬ ТА ТЕОРЕТИЧНІ ОСНОВИ

## 1.1 Основні поняття та визначення у сфері децентралізованих технологій

Децентралізовані технології — це широкий клас систем, які дозволяють розподіл обчислювальних та управлінських ресурсів серед різноманітних вузлів або агентів, а не зосереджують їх у одному центральному місці. Ця концепція передбачає, що кожен вузол у такій системі здатний незалежно обробляти дані та приймати рішення, що сприяє збільшенню надійності, безпеки та ефективності загалом.

Центральним елементом децентралізованих технологій є блокчейн — технологія розподілених реєстрів, яка дозволяє створювати безпечні, незмінні та прозорі системи обліку. Крім блокчейна, до сфери децентралізованих технологій також відносяться різні форми розподілених файлових систем, P2P-мережі та інші технології, що підтримують децентралізовану обробку та зберігання даних.

Блокчейн — це форма розподіленого реєстру технологій (DLT), яка забезпечує зберігання даних у ланцюгу блоків. Кожен блок у ланцюгу містить певну кількість транзакцій або записів, і кожен наступний блок містить хеш попереднього, створюючи незмінний та послідовний ланцюг. Ця структура гарантує, що один раз додані до блокчейна дані не можна змінити або видалити без зміни всіх наступних блоків, що робить систему надзвичайно безпечною.

У контексті дипломної роботи, яка фокусується на децентралізованих технологіях, особливо важливо розуміти різні види мереж та характеристики їх вузлів. Ось огляд основних типів мереж з акцентом на характеристики вузлів:

Централізовані Мережі

- Особливості Вузлів: В таких мережах є один або кілька центральних вузлів, які контролюють всю мережу. Ці вузли можуть бути серверами або центральними базами даних.

- Проблематика: Централізовані вузли створюють точки збою та збільшують ризик централізованих атак.

Децентралізовані Мережі

- Особливості Вузлів: У цьому типі мережі немає єдиного контролюючого центру. Кожен вузол працює незалежно, але є частиною загальної мережевої структури.

- Переваги: Зниження ризиків централізованого контролю та збоїв, а також підвищення стійкості до атак.

3. Розподілені Мережі (P2P - Peer-to-Peer)

- Особливості Вузлів: Кожен вузол (або "peer") в такій мережі діє як клієнт та сервер одночасно. Всі вузли мають однакові права та можливості [1].

- Застосування: Широко використовуються в системах файлообміну, криптовалютах та децентралізованих застосунках (DApps).

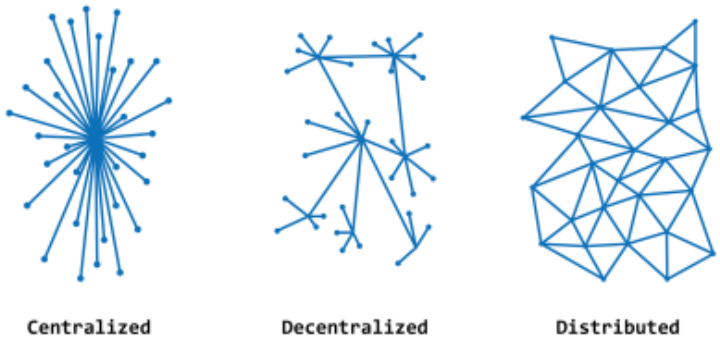



Рисунок 1.1 – Три підходи до побудови мереж

Блокчейн-Мережі

Вузли в блокчейн-мережах здійснюють валідацію та запис транзакцій в блоки. Можуть бути повноправними вузлами, які зберігають повну копію блокчейну, або легкими вузлами, які зберігають тільки частину даних.

- Специфіка: Вузли можуть брати участь у механізмах консенсусу, таких як Proof of Work або Proof of Stake, для підтримки безпеки та незмінності мережі.

Гібридні Мережі

Особливості Вузлів: Комбінують елементи централізованих та децентралізованих мереж. Вони можуть мати централізовані сервери для певних функцій та децентралізовані елементи для інших.

Переваги: Гібридні мережі можуть використовувати переваги обох систем, забезпечуючи гнучкість та масштабованість.

Основні характеристики блокчейну включають:

1. Децентралізація

В блокчейні немає центрального органу або адміністратора, який контролює мережу. Натомість, управління та обробка даних розподіляються серед усіх учасників мережі, які називаються вузлами. Кожен вузол має копію всієї бази даних блокчейна і працює на основі загальноприйнятих правил.

Демократичний підхід до управління: Цей підхід дозволяє досягти вищої ступеня справедливості і прозорості в процесі ведення записів і управління транзакціями. Оскільки ніхто не має повного контролю над мережею, важко зловживати системою або маніпулювати інформацією.

Консенсус як Основа Ухвалення Рішень: Для внесення змін у блокчейн потрібна згода більшості вузлів. Це забезпечує, що будь-які зміни, що вносяться в ланцюг, відображають загальну волю мережі, а не інтереси окремої групи чи особи.

- Відсутність Єдиної Точки Збою: Традиційні централізовані системи часто страждають від проблеми єдиної точки збою. Якщо центральний сервер або база даних зазнає збою, це може вивести з ладу всю систему. У блокчейні, оскільки кожен вузол мережі має повну копію всіх записів, вихід з ладу одного вузла не впливає на решту мережі.

Стійкість до Кібератак: Атакувати блокчейн значно складніше, ніж традиційні системи. Хакеру потрібно одночасно скомпрометувати велику кількість вузлів мережі для проведення успішної атаки, що є вкрай непростим завданням через розподілений характер технології.

Запобігання Маніпуляціям: Будь-які спроби маніпулювати даними в блокчейні вимагають надзвичайно великих обчислювальних ресурсів, оскільки потрібно змінити інформацію в кожному копії ланцюга на всіх вузлах мережі. Це робить фальсифікацію або маніпулювання майже неможливими.

Ці особливості блокчейна сприяють створенню міцної, безпечної та прозорої системи, яка ідеально підходить для децентралізованих застосунків, що вимагають високого рівня довіри і безпеки. У вашій дипломній роботі ви можете дослідити, як ці характеристики блокчейна впливають на розробку програмного забезпечення та як вони можуть бути використані для підвищення ефективності та безпеки різних систем.

2. Прозорість

- Відкритість Даних: У багатьох блокчейнах, особливо у публічних, кожна транзакція є публічною і може бути переглянута будь-яким учасником мережі. Це забезпечує високий рівень прозорості.

- Перевірка Історії Транзакцій: Кожен учасник має можливість перевірити історію транзакцій, що сприяє довірі та прозорості у мережі.

3. Незмінність

- Захист від Змін: Одного разу додана до блокчейна, інформація стає незмінною. Це означає, що транзакції не можна змінити або видалити, що забезпечує історію даних без можливості фальсифікації.

- Захист від Маніпуляцій: Щоб змінити інформацію в одному блоку, потрібно змінити всі наступні блоки, що вимагає величезних обчислювальних ресурсів і досягнення консенсусу в мережі, роблячи маніпуляції практично неможливими.

4. Консенсус

Механізми Консенсусу в Блокчейні

Механізми консенсусу є ключовою частиною блокчейн-технології. Вони використовуються для забезпечення одностайності у мережі щодо стану ланцюга блоків і допомагають вирішити, які блоки додаються до ланцюга.

1. Proof of Work (PoW)

В PoW, учасники (майнери) використовують свої обчислювальні ресурси для розв'язання складних математичних задач. Перший майнер, який знаходить рішення, отримує право додати новий блок до ланцюга. PoW є механізмом консенсусу, який використовується Bitcoin і багатьма іншими ранніми криптовалютами. Однак, цей процес може бути енергетично затратним, що викликає занепокоєння щодо його екологічного впливу.

2. Proof of Stake (PoS)

Підхід Proof of Stake (PoS) має кілька ключових відмінностей від традиційного механізму Proof of Work (PoW), який використовується у багатьох блокчейн-мережах, таких як Bitcoin.

Відмінності від PoW

В PoS, на відміну від PoW, де для майнінгу блоків використовуються обчислювальні ресурси (зокрема, велика обчислювальна потужність), валідатори

блоків визначаються на основі кількості їх володіння криптовалютою, що називається "стейком". Це означає, що учасники мережі, які володіють більшою кількістю токенів або монет, мають більшу ймовірність бути обраними для створення нового блоку та отримання винагороди за транзакції.

Менший Енергетичний Вплив

PoS вважається більш енергоефективним порівняно з PoW, оскільки він не потребує використання масивних обчислювальних ферм, які споживають великі об'єми електроенергії для вирішення складних математичних задач. Відповідно, PoS значно знижує енергетичний вплив блокчейн-мереж, роблячи їх більш сталими та екологічними.

В системі PoS, валідатори мотивовані підтримувати чесність та безпеку мережі, оскільки за шахрайську поведінку або зловмисні дії вони можуть втратити свій стейк. Це створює фінансовий стимул для валідаторів діяти в інтересах мережі, а не намагатися обдурити систему. Такий підхід забезпечує високий рівень безпеки, оскільки потенційні втрати від шахрайства зазвичай значно перевищують потенційну користь.

Загалом, PoS пропонує більш сталу та ефективну альтернативу PoW, зосереджуючись на зменшенні енергетичного впливу, забезпеченні безпеки мережі та створенні стимулів для чесної поведінки учасників мережі.

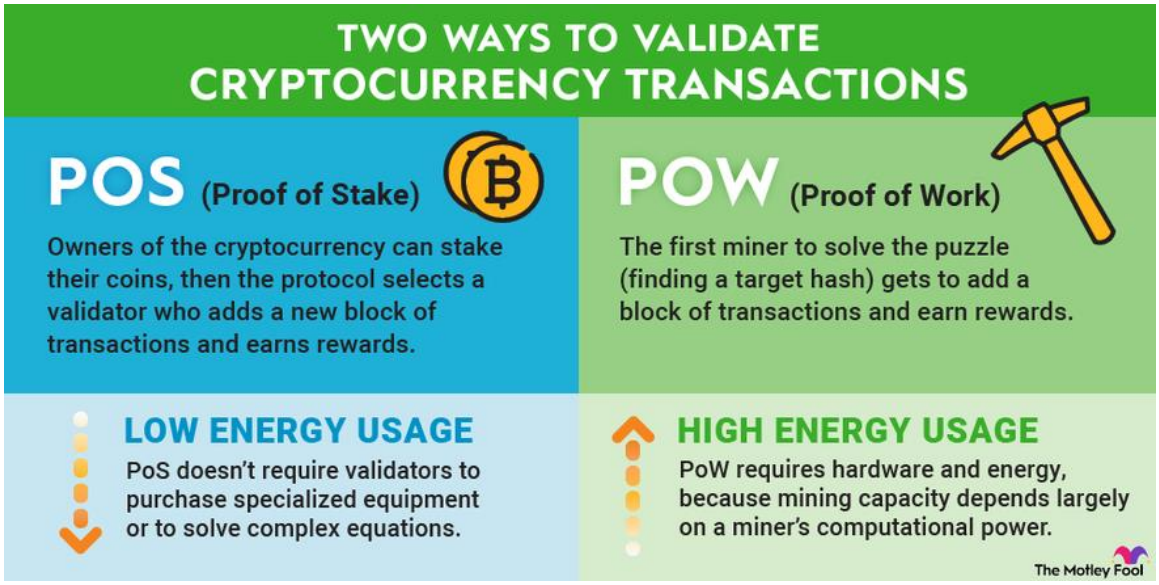



Рисунок 1.2 – Відмінності в алгоритмах консенсусу

Валідація та Безпека

- *Забезпечення Чесності*: Механізми консенсусу забезпечують, що учасники мережі діють чесно. Вони знижують ймовірність шахрайства, оскільки маніпулювати системою економічно невигідно або технічно нездійсненно.

- *Підтримка Цілісності Мережі*: Ці механізми важливі для підтримки цілісності і незмінності блокчейну, гарантуючи, що всі блоки, які додаються до ланцюга, є валідними та відображають реальні транзакції.

Загальний Вплив

- *Вибір Механізму Консенсусу*: Вибір між PoW, PoS, та іншими методами залежить від конкретних вимог та цілей блокчейн-проекту. Кожен механізм має свої переваги та недоліки, в тому числі різні рівні безпеки, швидкості та енергетичної ефективності.

- *Забезпечення Розвитку та Адаптації*: В майбутньому можливий розвиток та адаптація нових форм консенсусу, які можуть пропонувати покращену безпеку, зменшене енергоспоживання та підвищену масштабованість.

Блокчейн має широкий спектр застосувань, включаючи криптовалюти (наприклад, Bitcoin), смарт-контракти (наприклад, Ethereum), ланцюжки постачань, цифрову ідентифікацію, виборчі системи та багато іншого. Він забезпечує надійність, прозорість та безпеку у системах, де ці атрибути мають критичне значення.

## 2.2 Огляд Веб-Технологій у контексті децентралізації

У контексті ширшого дослідження децентралізованих технологій, підпункт "Огляд веб-технологій у контексті децентралізації" зосереджується на аналізі ролі та впливу веб-технологій у створенні та розвитку децентралізованих систем. Цей аспект є особливо значущим у світлі сучасних інноваційних трендів у сфері IT, де децентралізація відіграє ключову роль у формуванні нових підходів до розробки, розгортання та використання веб-додатків і сервісів.

Для детального розширення на тему "Історичний Контекст та Еволюція Веб-Технологій і Вплив Децентралізації", можна розглянути наступні аспекти:

Виникнення World Wide Web (WWW) в 1989 році, яке започаткував Тім Бернерс-Лі, стало фундаментальним моментом у розвитку сучасного інтернету. Концепція, розроблена Бернерс-Лі, полягала у створенні системи, що дозволяє з'єднувати документи та інші ресурси за допомогою гіперпосилань. Це стало ключовим елементом, який визначив основну структуру та спосіб взаємодії у глобальній інформаційній мережі. WWW дозволило створити зв'язану мережу документів, доступних через інтернет, що суттєво змінило спосіб, яким люди збирають, обмінюються та використовують інформацію.

Ранні веб-сайти, що з'явилися в рамках цієї новоствореної мережі, були переважно статичними. Це означає, що вони надавали інформацію в односторонньому порядку, без можливості інтерактивної взаємодії з користувачем. Статичні веб-сайти складалися з HTML-сторінок, які зберігались на сервері і передавались в незмінному вигляді до веб-браузера користувача. Ці ранні веб-сайти не мали можливостей динамічного контенту чи користувацьких інтеракцій, таких як форми для введення даних чи інтерактивні елементи, які відповідають на дії користувача.

Ці два фундаментальні аспекти - створення WWW Тімом Бернерс-Лі та розвиток перших веб-сайтів - стали основою для подальшого еволюційного розвитку інтернету. Вони заклали основи для багатьох інновацій, які згодом з'явилися у веб-технологіях, та сприяли створенню глобальної інформаційної мережі, що пов'язує мільярди людей по всьому світу.

HTML та HTTP.

HTML, який стоїть за HyperText Markup Language, є фундаментальною технологією, що лежить в основі вебу. Як мова розмітки, HTML використовується для структурування та організації контенту на веб-сторінках. Його основна роль полягає в тому, щоб вказати веб-браузерам, як відображати текст, зображення, посилання, таблиці, списки та інші елементи на веб-сторінці. HTML складається з ряду елементів, які огортають різні частини контенту, надаючи їм семантичне значення та інструкції щодо відображення. Кожен елемент описується за допомогою тегів, і ця структура дозволяє браузерам інтерпретувати та відображати контент відповідно до намірів розробника.

HTTP, або HyperText Transfer Protocol, є основним протоколом для передачі

даних у Інтернеті. HTTP визначає спосіб, яким веб-клієнти (зазвичай веб-браузери) та веб-сервери спілкуються між собою. Коли користувач вводить URL в браузері або натискає на посилання, відбувається HTTP-запит до веб-сервера. Веб-сервер обробляє запит і відправляє відповідь назад до клієнта, яка зазвичай містить HTML-документ, а також може включати стилі CSS, JavaScript-скрипти, зображення та інші типи медіа. HTTP дозволив створити динамічний та інтерактивний Інтернет, де користувачі можуть безперервно переходити від одного ресурсу до іншого, отримуючи доступ до широкого спектру веб-контенту.

HTML та HTTP разом становлять основу вебу, як ми його знаємо сьогодні. HTML забезпечує структуру та зміст веб-сторінок, тоді як HTTP дозволяє браузерам запитувати та отримувати ці сторінки. Ці технології визначають, як веб-сайти створюються, як вони взаємодіють і як користувачі взаємодіють із цими сайтами.

JavaScript і AJAX

Інтеграція JavaScript у веб-сайти відіграла значну роль у розвитку інтернету, перетворюючи статичні веб-сторінки на динамічні та інтерактивні веб-додатки. JavaScript є мовою програмування високого рівня, яка дозволяє виконувати складні операції на стороні клієнта, в браузері користувача, без необхідності постійної взаємодії з сервером. Це включає в себе зміну вмісту сторінки, анімації, контроль за формами, відповідь на дії користувача, такі як кліки мишею або натискання клавіш, та багато іншого. Це дало можливість розробникам створювати веб-сайти, які активно реагують на дії користувача, покращуючи їх взаємодію та забезпечуючи більш плавний та привабливий користувацький досвід.

AJAX (Asynchronous JavaScript and XML) – це техніка, яка була запроваджена у 2000-х роках і стала ключовим компонентом для створення високоінтерактивних веб-додатків. AJAX дозволяє веб-додаткам асинхронно надсилати та отримувати дані з сервера без потреби перезавантажувати всю сторінку. Це означає, що частина даних на веб-сторінці може бути оновлена незалежно від інших, що забезпечує швидку та ефективну взаємодію з користувачем. Завдяки AJAX, веб-додатки стали швидшими та зручнішими, оскільки значно скоротилася кількість часу, необхідного для завантаження та оновлення вмісту сторінки.

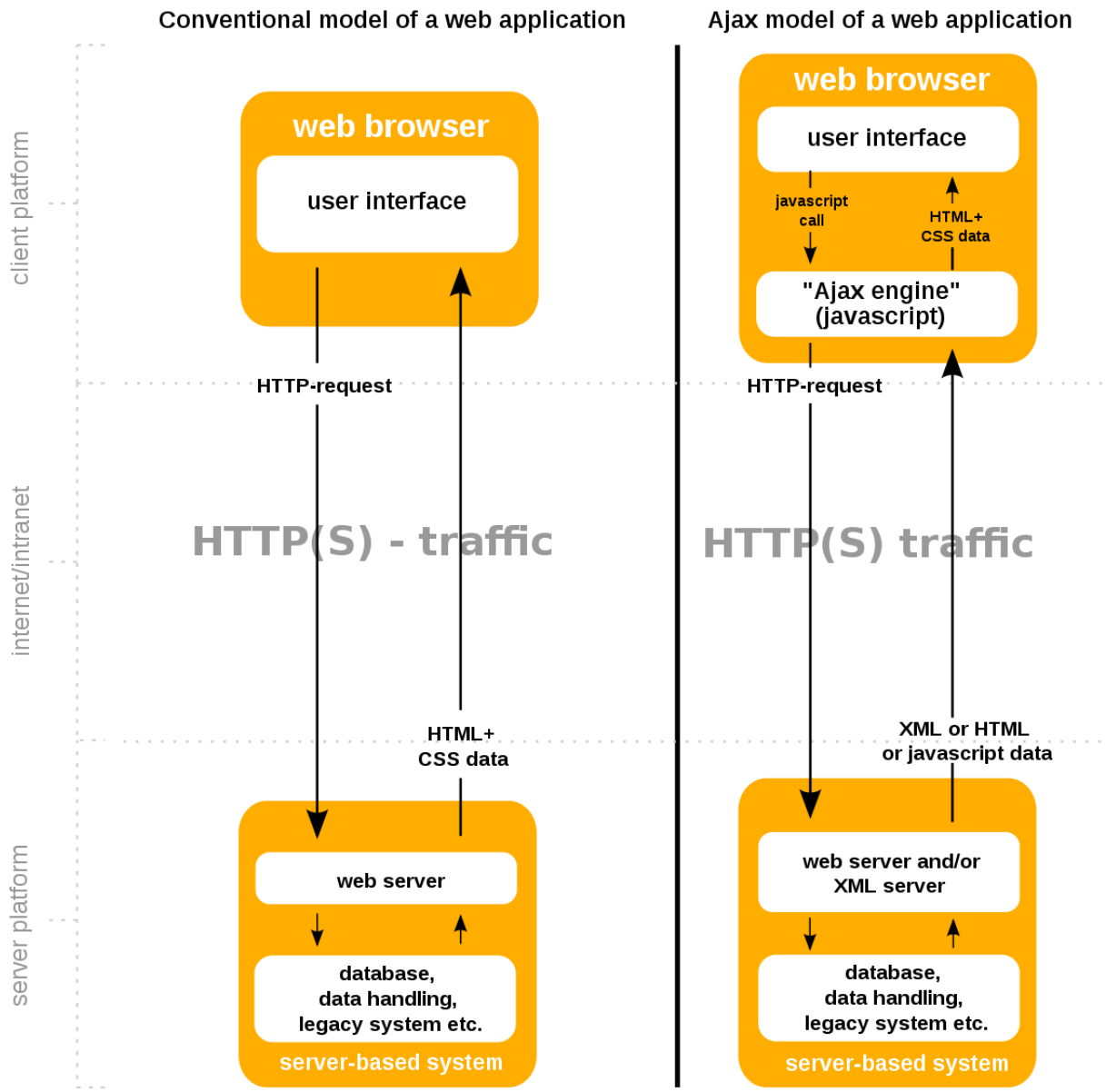



Рисунок 1.3 – Візуалізація роботи AJAX-запитів

Впровадження JavaScript та AJAX в сучасні веб-технології змінило парадигму веб-розробки, перетворивши її з простого показу статичного вмісту на створення багатих, динамічних та інтерактивних веб-досвідів. Ці технології підсилили можливості веб-додатків, зробивши їх схожими на традиційні програми для робочих столів за своєю функціональністю та взаємодією з користувачем.

Вплив Веб 2.0

Соціальні Мережі та Інтерактивність

- Демократизація Інтернету: Ера Веб 2.0 ознаменувалася появою соціальних

мереж, блогів та вікі-сайтів, які зробили інтернет більш відкритим та доступним для користувачів, сприяючи спільному творчеству та спілкуванню.

Клієнтська Сторона та Фреймворки

- Фронтенд Фреймворки: Розвиток фронтенд технологій, включаючи фреймворки та бібліотеки на стороні клієнта, такі як React.js, Angular, Vue.js, змінив підхід до розробки веб-інтерфейсів, дозволяючи створювати складні та інтуїтивні користувацькі інтерфейси. Фреймворк у сфері програмування та розробки програмного забезпечення - це стандартизована платформа, яка визначає основу для розробки і запуску програмних додатків. Він надає готову архітектуру та набір попередньо написаних компонентів та бібліотек, що дозволяє розробникам зосередитися на унікальних аспектах своїх проектів, замість того, щоб витрачати час на написання базового коду з нуля.

Фреймворки призначені для спрощення процесу розробки, надаючи стандартні шаблони та засоби для реалізації загальних завдань, таких як управління даними, взаємодія з базами даних, рендеринг інтерфейсу користувача та забезпечення взаємодії між різними компонентами програми. Це допомагає розробникам уникнути повторного кодування загальних функцій та фокусуватися на специфіці своїх проектів, підвищуючи продуктивність та ефективність розробки.

## 1.2 Вплив Децентралізації на Веб-Технології

- Блокчейн та Розподілені Технології: Впровадження децентралізованих технологій, особливо блокчейну, почало змінювати парадигму веб-розвитку, надаючи нові можливості для створення розподілених, прозорих та безпечних веб-додатків.

- Децентралізовані Додатки (DApps): Розвиток децентралізованих додатків надає користувачам більше контролю над їхніми даними та відкриває нові шляхи для інновацій в області захисту приватності та безпеки даних.

Цей підпункт надасть глибокий та всебічний аналіз веб-технологій в ері децентралізації, демонструючи їх роль у сучасному цифровому ландшафті та їх

вплив на майбутнє веб-розробки.

- Злам Парадигми Централізованих Систем: Поява блокчейн-технологій та децентралізованих систем в середині 2010-х років почала змінювати традиційний підхід до розробки та використання веб-додатків.
- Блокчейн та Децентралізовані Додатки (DApps): Завдяки блокчейну з'явилась можливість створювати децентралізовані додатки, які не залежать від єдиного центрального сервера та пропонують новий рівень безпеки, прозорості та відсутності посередників.
- Смарт-Контракти та Автоматизація: Смарт-контракти, особливо у мережах, таких як Ethereum, забезпечили новий спосіб автоматизації та взаємодії в рамках децентралізованих систем.

Сучасний Стан та Перспективи

- Інтеграція та Розвиток: Постійний розвиток інструментів, які інтегрують децентралізацію в традиційні веб-технології, відкриває нові можливості для розробників та користувачів.

- Виклики та Можливості: В той час як децентралізація пропонує значні переваги, існують також виклики, пов'язані з масштабованістю, швидкістю та прийняттям нових технологій.

Цей історичний огляд веб-технологій та їхньої еволюції у контексті децентралізації підкреслює динамічний та інноваційний характер сфери веб-розробки. Він також вказує на значний потенціал децентралізації у формуванні майбутнього цифрового ландшафту.

## 1.3 Роль Децентралізації у Веб-Розробці

Вплив на Архітектуру Веб-Додатків

1. Модель Розподіленої Архітектури:

- Традиційні веб-додатки зазвичай залежать від централізованого сервера або кластера серверів, що обробляють запити користувачів. Децентралізація вносить парадигму розподіленої архітектури, де функції, такі як обробка даних,

автентифікація, зберігання даних, виконуються через мережу різних вузлів.

- Це знижує залежність від одного сервера або сервісу, підвищуючи надійність і доступність.

2. Використання Blockchain і DApps:

Блокчейн Технології та Розробка Децентралізованих Додатків (DApps)

Основи Блокчейн Технологій

Блокчейн є технологією розподілених реєстрів, де кожен блок даних (що містить транзакції) з'єднаний з попереднім блоком через криптографічний хеш.

- Ця структура створює незмінний ланцюг блоків, забезпечуючи надійність і прозорість історії транзакцій.

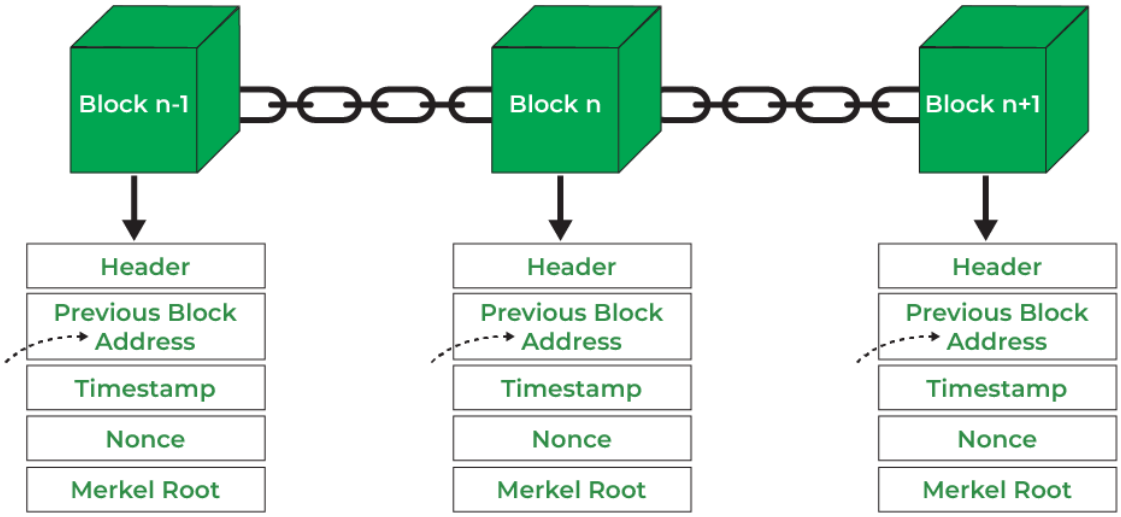



Рисунок 1.4 – Структура та елементи структури даних блокчейн

2. Децентралізація:

- В блокчейні немає центральної контрольної організації. Кожен учасник мережі має однакові права і можливості у валідації та верифікації транзакцій.

Ethereum і Розвиток DApps

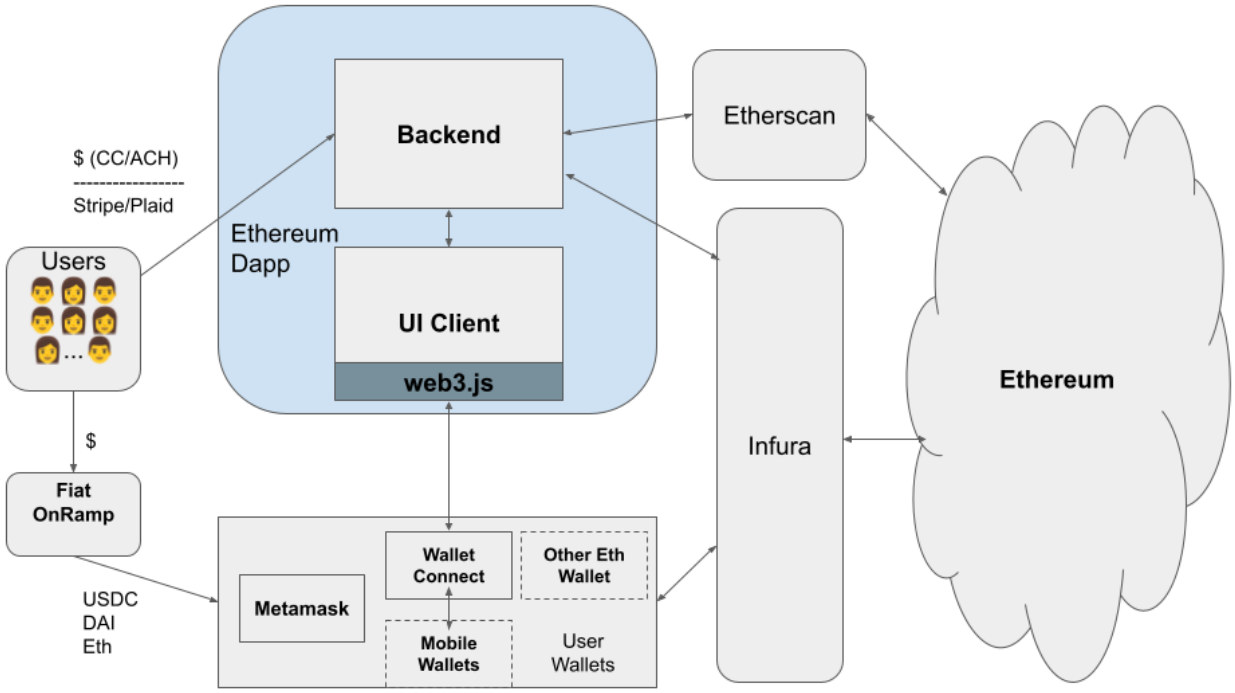



Рисунок 1.4 – Архітектура децентралізованого додатку для мережі Ethereum

1. Ethereum як Платформа:

- Ethereum розширює концепцію блокчейну, включаючи функціональність смарт-контрактів, яка перетворює його на повноцінну платформу для створення додатків.

2. Смарт-Контракти:

Смарт-контракти представляють собою одну з ключових інновацій у сфері блокчейн-технологій. Вони є програмами, які засновані на блокчейні і призначені для автоматичного виконання попередньо заданих умов контракту. Ця технологія дозволяє забезпечити виконання домовленостей і угод без потреби у залученні третіх сторін або посередників, таких як юристи або банки. Це важливо, оскільки смарт-контракти забезпечують високий рівень прозорості, безпеки та надійності у фінансових та інших угодах.

Смарт-контракти Ethereum є одними з найбільш популярних і використовуваних видів смарт-контрактів. Вони написані на мові програмування Solidity, яка спеціально розроблена для створення смарт-контрактів на Ethereum. Solidity - це об'єктно-орієнтована, високорівнева мова програмування, яка дозволяє

розробникам формулювати складні умови та логіку, які потім впроваджуються і автоматично виконуються в блокчейні Ethereum. Крім Solidity, смарт-контракти Ethereum можуть бути написані на інших сумісних мовах, але Solidity залишається найпопулярнішою через свою спеціалізацію і зручність використання.

```
contract token {
    mapping (address => uint) public coinBalanceOf;
    event CoinTransfer(address sender, address receiver, uint amount);

  /* Initializes contract with initial supply tokens to the creator of the contract */
  function token(uint supply) {
        if (supply == 0) supply = 10000;
        coinBalanceOf[msg.sender] = supply;
    }

  /* Very simple trade function */
    function sendCoin(address receiver, uint amount) returns(bool sufficient) {
        if (coinBalanceOf[msg.sender] < amount) return false;
        coinBalanceOf[msg.sender] -= amount;
        coinBalanceOf[receiver] += amount;
        CoinTransfer(msg.sender, receiver, amount);
        return true;
    }
}
```

Рисунок 1.5 – Приклад коду смарт-контракту на Solidity

Смарт-контракти революціонізували підхід до заключення та виконання контрактів, зокрема в сфері фінансів, нерухомості, юридичних послуг та багатьох інших галузях. Вони автоматизують процеси, знижують ризики помилок або шахрайства, а також зменшують час і витрати на виконання угод. Смарт-контракти стають все більш популярними у світі блокчейну та вважаються однією з найбільш перспективних технологій у цій області.

3. Розвиток DApps:

- DApps (децентралізовані додатки) використовують блокчейн для реалізації своєї функціональності. Вони можуть використовувати Ethereum для управління транзакціями, зберігання даних, взаємодії з смарт-контрактами та інших функцій.

- Такі додатки не залежать від централізованих серверів, що забезпечує більш високий рівень безпеки, прозорості та стійкості до цензури.

Переваги та Виклики

1. Переваги:

- Прозорість та Безпека: Всі транзакції перевіряються учасниками мережі, що забезпечує високу прозорість.

- Відсутність Посередників: Смарт-контракти дозволяють угодам автоматично виконуватися, усуваючи потребу у посередниках.

2. Виклики:

- Масштабованість: Через обмеження кількості транзакцій, які можуть бути оброблені за одиницю часу, існують питання з масштабованістю.

- Складність Розробки: Розробка на Ethereum вимагає знань у певних мовах програмування та розуміння принципів блокчейну.

Розвиток технологій Ethereum та створення на їх базі DApps значно змінює пейзаж веб-розробки, відкриваючи нові можливості для інновацій та створення більш безпечних та ефективних додатків. Це є важливою частиною дослідження у контексті дипломної роботи, оскільки вказує на майбутні напрямки розвитку веб-технологій.

Вплив на Функціональність

1. Безпека:

- Децентралізація забезпечує вищий рівень безпеки, оскільки відсутність центрального контрольного пункту ускладнює хакерські атаки.

- Розподілена природа даних в блокчейні означає, що для зміни чи фальсифікації інформації потрібно одночасно змінити її на всіх вузлах, що практично неможливо.

2. Приватність:

- Децентралізовані системи можуть забезпечити кращий захист приватності користувачів, оскільки дані не зберігаються на централізованому сервері, де вони можуть бути легко доступні третім сторонам.

3. Доступність та Надійність:

- Відсутність єдиної точки збою робить децентралізовані веб-додатки більш стійкими до відмов. Якщо один вузол виходить з ладу, інші продовжують працювати, забезпечуючи неперервність служб.

4. Ефективність:

- Хоча децентралізація може підвищувати ефективність за рахунок розподіленої обробки, існують виклики, пов'язані зі швидкістю транзакцій та масштабованістю, особливо в блокчейн-мережах, де час обробки блоку може впливати на продуктивність.

Децентралізація відкриває нові горизонти в розробці веб-додатків, пропонуючи альтернативні підходи до зберігання даних, безпеки, приватності та надійності. Однак, вона також приносить нові виклики, такі як забезпечення швидкості та ефективності обробки даних. Ваше дослідження у цій області може виявити значний потенціал децентралізації для вирішення існуючих проблем веб-розробки, а також сприяти інноваціям у створенні нових типів веб-додатків.

## 1.4 Технології та інструменти в децентралізованих веб-додатках

. Підключення до Блокчейн API та Web3:

- Використання бібліотек, таких як Web3.js, дозволяє React-компонентам взаємодіяти з Ethereum блокчейном, здійснюючи виклики до смарт-контрактів, обробляючи транзакції та виконуючи інші операції.

Web3.js є популярною JavaScript бібліотекою, яка використовується для взаємодії з Ethereum блокчейном. Ця бібліотека дозволяє веб-додаткам (зазвичай виконуваним у веб-браузері) взаємодіяти з Ethereum блокчейном, зокрема відправляти транзакції, взаємодіяти зі смарт-контрактами та читати блокчейн-дані.

Взаємодія з Ethereum блокчейном: Web3.js служить як міст між Ethereum блокчейном та веб-додатком. Це дозволяє веб-додаткам виконувати функції, схожі на ті, які виконуються в Ethereum валетах та інших блокчейн-додатках.

Робота з Смарт-контрактами: Однією з ключових функцій Web3.js є можливість взаємодії зі смарт-контрактами, розгорнутими на Ethereum блокчейні. Розробники можуть використовувати Web3.js для відправлення транзакцій до смарт-контрактів, викликати їхні функції та обробляти події, які вони випускають.

Підключення до Ethereum вузлів: Web3.js використовує Ethereum вузли для відправлення та отримання інформації від блокчейну. Вона може підключатися до локальних вузлів або вузлів, наданих послугами третіх сторін, таких як Infura.

Розробники можуть використовувати Web3.js для створення різноманітних

децентралізованих додатків (DApps), від фінансових додатків та ігор до систем голосування та торговельних платформ. Ця бібліотека є незамінною для будь-якого, хто хоче взаємодіяти з Ethereum блокчейном через веб-додаток.

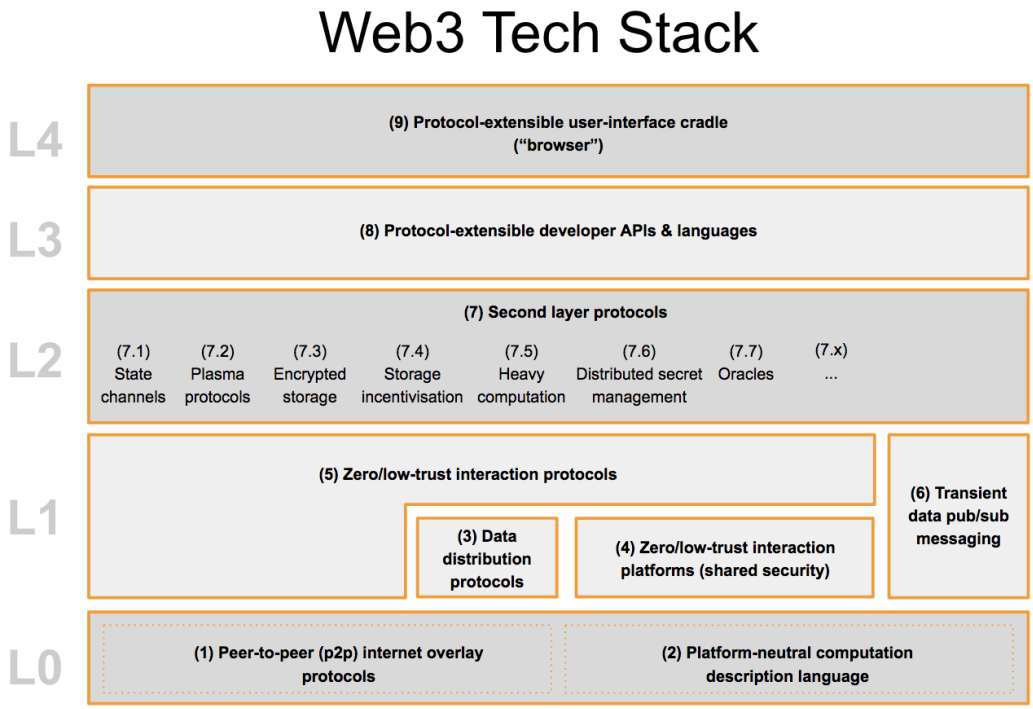



Рисунок 1.7 – Елементи стеку технологій для Web3

Вплив на Веб-Розробку

Децентралізовані Веб-Додатки (DApps):

- Блокчейн відкриває можливості для створення децентралізованих веб-додатків, які не залежать від централізованої інфраструктури. DApps можуть пропонувати новий рівень безпеки, прозорості та функціональності.

2. Смарт-Контракти:

- Смарт-контракти - це самовиконуючі контракти з умовами угоди прямо закодованими в коді. Вони автоматизують виконання контрактів, зменшують потребу у посередниках та забезпечують високий рівень безпеки.

- Використання смарт-контрактів в веб-додатках може суттєво змінити спосіб, яким обробляються транзакції, управління даними, аутентифікація користувачів та багато іншого.

3. Децентралізовані Бази Даних:

- Блокчейн дозволяє створювати децентралізовані бази даних, що можуть

бути використані для безпечного зберігання та обміну даними. Це відкриває нові можливості для веб-додатків, що потребують високого рівня надійності даних.

Технологічні Імплікації

Зміна Моделі Безпеки:

- Блокчейн вносить зміни в традиційні моделі безпеки, оскільки кожна транзакція перевіряється мережею і зашифровується. Це забезпечує вищий рівень безпеки в порівнянні з традиційними веб-додатками.

Взаємодія з Веб-Інтерфейсами:

- Інтеграція блокчейну з веб-додатками часто вимагає спеціальних бібліотек та API, таких як Web3.js, що дозволяють веб-додаткам взаємодіяти з блокчейнами, наприклад, Ethereum.

Питання Масштабованості та Продуктивності:

- Блокчейн технології мають обмеження щодо масштабованості та швидкості транзакцій, що може впливати на продуктивність децентралізованих веб-додатків.

Вплив блокчейну на веб-технології є значним і багатогранним, пропонуючи нові можливості для безпеки, прозорості та інновацій у веб-додатках. Розробка в цій сфері вимагає глибокого розуміння блокчейн-технологій, а також вміння адаптувати традиційні веб-практики для інтеграції з цими новими, динамічними системами.

## 1.5 Успішні приклади децентралізованих веб-додатків

Ethereum і Його Роль у Розвитку DApps

Ethereum представляє собою відкриту блокчейн-платформу, яка розширює можливості традиційного блокчейну, надаючи розробникам інструменти для створення та запуску децентралізованих додатків, відомих як DApps. Ця платформа знаменита своєю унікальною характеристикою – здатністю виконувати смарт-контракти. Ці смарт-контракти є програмами, які автоматизують виконання контрактів, забезпечуючи виконання угод за допомогою коду без потреби у посередниках. Це дозволяє автоматизувати різноманітні процеси та транзакції, значно збільшуючи ефективність та надійність.

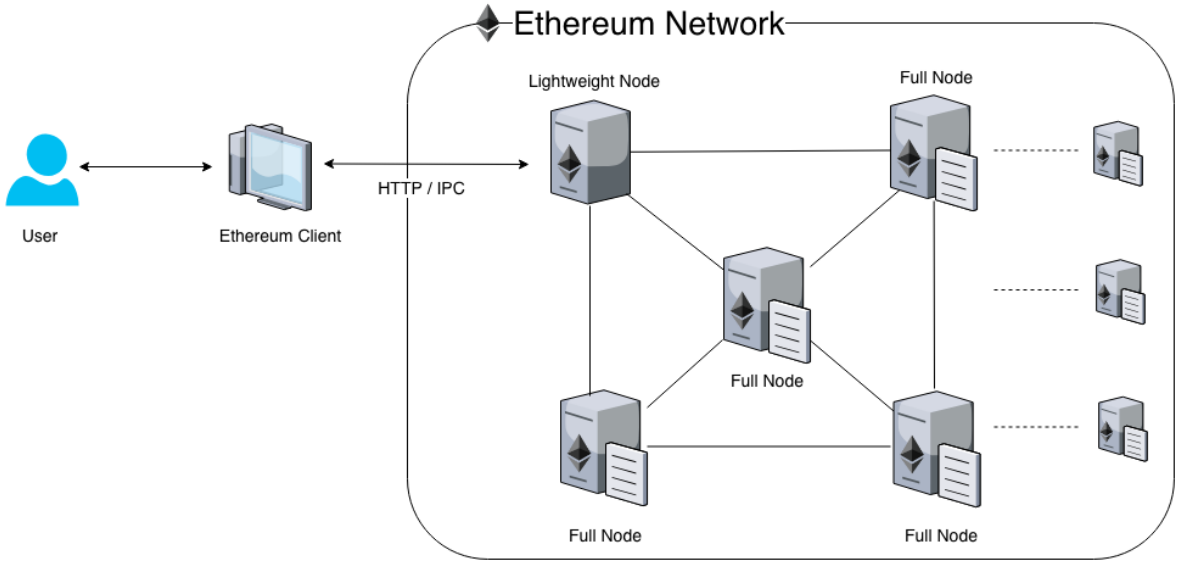



Рисунок 1.8 – Спрощена структура мережі Ethereum та взаємодії користувача з нею

Одним із ключових аспектів Ethereum є використання мови програмування Solidity для створення смарт-контрактів. Solidity спеціально розроблена для блокчейн-платформ, як Ethereum, і дозволяє розробникам створювати складні смарт-контракти з високим рівнем гнучкості та функціональності. Ця мова програмування дозволяє розробникам втілювати складні алгоритми і бізнес-логіку безпосередньо у блокчейн, що відкриває широкі можливості для інновацій у розробці DApps.

Екосистема Ethereum:

Екосистема Ethereum – це розгалужена та багатогранна мережа, яка об'єднує мільйони користувачів по всьому світу. Вона включає в себе не лише розробників, які створюють децентралізовані додатки на основі цієї платформи, але й майнерів, які підтримують її роботу та безпеку через процес валідації транзакцій, а також інвесторів, що вкладають кошти в проекти, засновані на Ethereum.

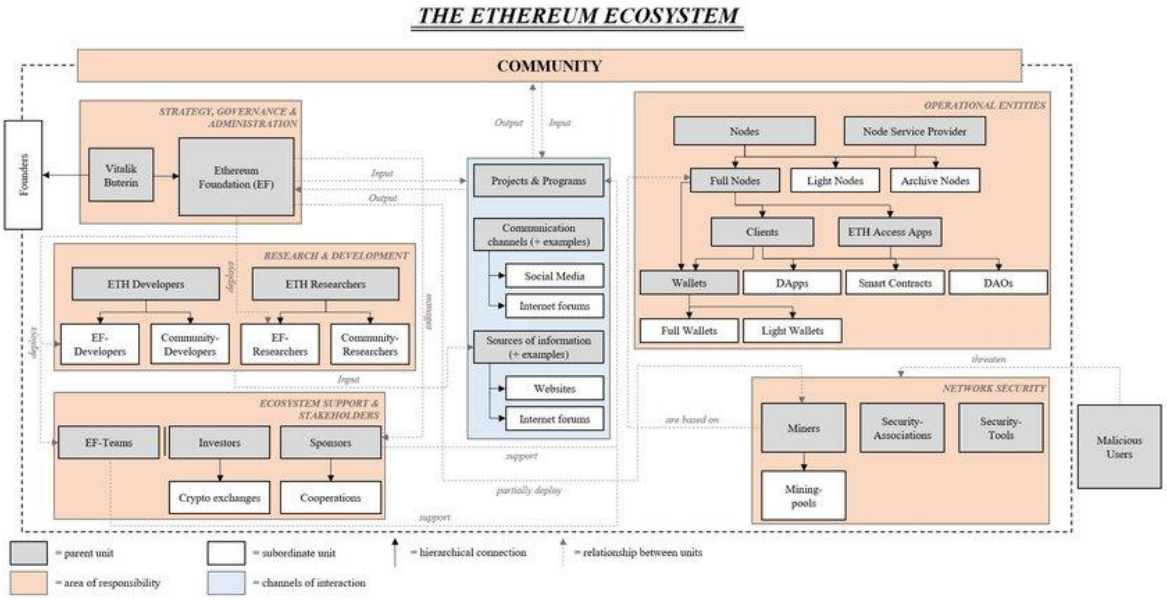



Рисунок 1.9 – Екосистема мережі Ethereum

Ця екосистема відрізняється своєю динамічністю та інноваційністю, оскільки пропонує широкий спектр можливостей для застосування технології блокчейн у різних сферах. Від фінансових послуг, таких як криптовалютні транзакції та децентралізоване фінансування (DeFi), до галузі розваг, включаючи ігри на блокчейні та цифрові колекції, екосистема Ethereum стимулює інновації та новаторські підходи.

Крім того, Ethereum має значний вплив на соціальні медіа та інші форми цифрової комунікації, де використання децентралізованих додатків і смарт-контрактів відкриває нові шляхи для створення довіри, прозорості та безпеки в онлайн-взаємодіях. Учасники цієї екосистеми не лише споживають та вкладають у технологію, але й активно формують її майбутнє, вносячи інновації та вдосконалення, що сприяє її еволюції та розширенню можливостей використання.

Приклад Застосування: MakerDAO

MakerDAO являє собою важливий елемент екосистеми децентралізованого фінансування (DeFi), розташованої на платформі Ethereum. Ця платформа відрізняється тим, що дозволяє користувачам отримувати позики у власній стабільній валюті DAI, яка забезпечена криптовалютою. Ключовим аспектом MakerDAO є можливість закладати криптовалюту, таку як Ether, в якості застави для отримання DAI. Цей механізм дозволяє користувачам знизити ризики, пов'язані

з волатильністю криптовалют, а також отримати ліквідність, не продаваючи свої активи.

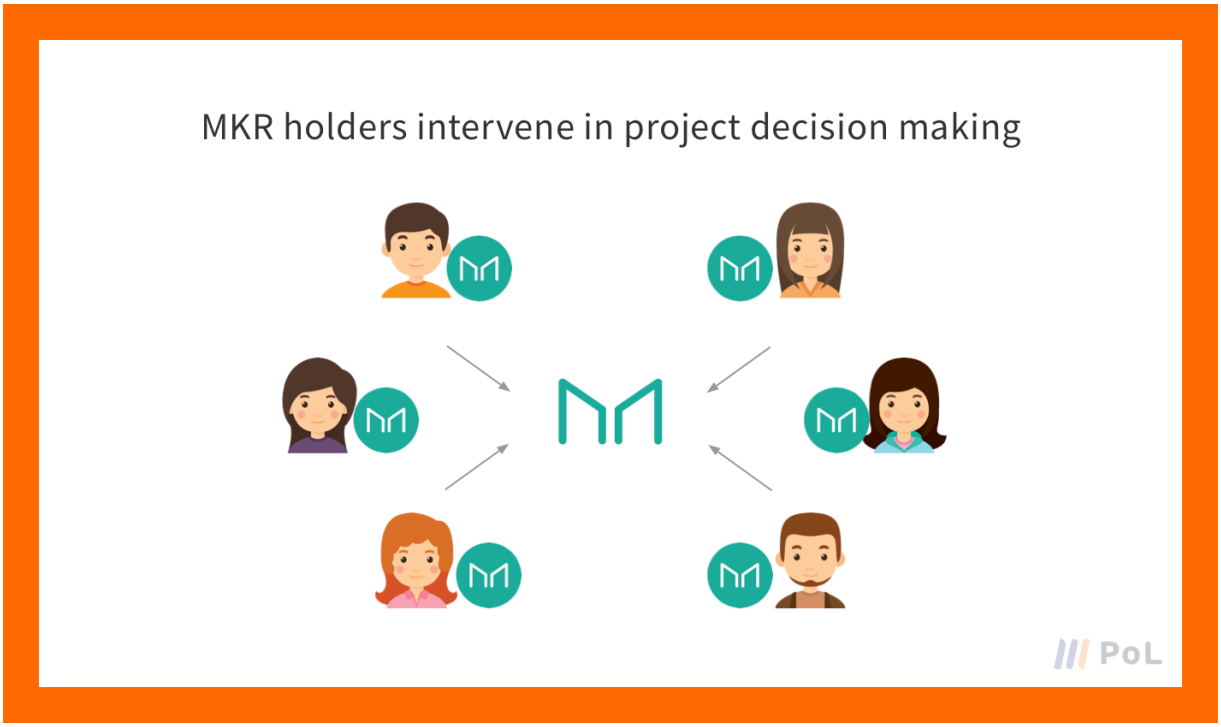



Рисунок 1.10 – Модальне вікно
додавання нової транзакції

Однією з основних переваг MakerDAO є сприяння фінансовій інклюзивності. Платформа відкриває доступ до фінансових послуг для більш широкого кола людей, не залежно від їх географічного розташування чи кредитної історії. Це робить MakerDAO важливим інструментом для розширення можливостей в сфері фінансів, особливо в умовах, коли традиційні банківські системи можуть бути недоступними або обмеженими для певних категорій населення.

Крім того, MakerDAO забезпечує високий рівень прозорості та безпеки. Всі угоди та умови кредитування зберігаються в блокчейні, що гарантує їх незмінність та прозорість. Це важливо для підвищення довіри користувачів до платформи, а також для забезпечення безпеки фінансових транзакцій. Такий підхід дозволяє MakerDAO стати надійним та привабливим вибором для тих, хто шукає альтернативні способи фінансування та управління своїми активами у світі криптовалют.

Ethereum як лідер у створенні децентралізованих додатків відіграє ключову

роль у розвитку та популяризації DeFi та інших DApps. Платформи, такі як MakerDAO, демонструють потенціал Ethereum у створенні нових фінансових систем, які є більш доступними, прозорими та безпечними, ніж традиційні фінансові інституції. Використання смарт-контрактів і децентралізованої структури забезпечує інноваційний підхід до розв'язання сучасних фінансових та соціальних викликів.

Steemit:

- Опис: Steemit – це блокчейн-базована соціальна медіа платформа, де користувачі отримують винагороду криптовалютою за створення та курування контенту.

- Переваги: Винагорода за якісний контент, децентралізоване управління та відсутність централізованої цензури.

Decentraland:

Decentraland представляє собою інноваційний віртуальний світ, де користувачі мають можливість не тільки купувати та продавати земельні ділянки, але й активно розвивати їх. Цей світ є унікальною комбінацією віртуальної реальності та блокчейн-технології, де основою для управління власністю та здійснення транзакцій виступає платформа Ethereum. Використання Ethereum в Decentraland дозволяє користувачам не лише керувати своїми активами на прозорий та безпечний спосіб, але й гарантує децентралізоване володіння цими активами.

Однією з ключових переваг Decentraland є можливість для користувачів створювати власний, індивідуальний контент на своїх земельних ділянках. Це створює надзвичайно гнучке та творче середовище, де користувачі можуть втілювати свої ідеї у віртуальному просторі, створюючи унікальні віртуальні витвори, будівлі та навіть цілі спільноти. Така модель сприяє формуванню багатогранної віртуальної екосистеми, де кожен учасник може вносити свій внесок у розвиток та дизайн віртуального світу.

Використання блокчейну в Decentraland гарантує, що всі транзакції та володіння земельними ділянками є повністю прозорими та незмінними. Це означає, що кожен користувач має незаперечне право на власність своїх активів, що робить віртуальний світ Decentraland надійним та безпечним місцем для інвестицій та

творчості.

Крім того, Decentraland пропонує різноманітні можливості для спільноти, включаючи соціальні заходи, ігри та інші види взаємодій, що робить його не просто платформою для управління віртуальною нерухомістю, але й живим, динамічним середовищем для розвитку віртуальних спільнот.

### 1.6 Виклики децентралізованих веб-додатків

Масштабованість у Контексті Децентралізованих Додатків (DApps)

Ключові Аспекти:

- Масштабованість у контексті DApps визначається як здатність системи адекватно впоратися зі збільшенням обсягу транзакцій та кількості активних користувачів. Це включає в себе підтримку високого рівня продуктивності, швидкості обробки транзакцій та ефективності інтерфейсу користувача при збільшенні навантаження на систему.

Транзакційна Пропускна Спроможність:

- Одним з ключових елементів масштабованості є пропускна спроможність мережі, тобто кількість транзакцій, які система може обробити за одиницю часу. Для DApps, які функціонують на блокчейн-платформах, це часто обмежується характеристиками самої блокчейн-платформи.

Швидкість Обробки Транзакцій:

- Ефективна масштабованість також означає, що час обробки транзакцій повинен залишатися прийнятним, незалежно від загального обсягу транзакцій в мережі.

Технічні Виклики Масштабованості

Масштабованість є одним з найбільших викликів для DApps, що обмежує їх потенціал широкого прийняття та використання. Розв'язання проблем масштабованості вимагає інноваційних технічних рішень та підходів, які можуть включати шардінг, Layer 2 рішення та оптимізацію смарт-контрактів. Успішне впровадження цих стратегій може значно підвищити продуктивність та ефективність DApps.

Обмеження Платформи:

- Багато блокчейн-платформ, особливо ті, які використовують PoW, мають обмеження в термінах транзакцій на секунду (TPS), що впливає на здатність DApps масштабуватися.

Проблеми Зберігання Даних:

- Збільшення обсягу транзакцій та користувачів може призвести до проблем зберігання даних, оскільки кожен вузол блокчейну зберігає повну копію ланцюга блоків.

Вартість Транзакцій:

- Під час високих піків навантаження на мережу вартість транзакцій (газ у випадку Ethereum) може значно зрости, що робить використання DApps дорогим.

Стратегії Покращення Масштабованості

Шардінг:

- Шардінг є ключовою концепцією в блокчейн-технологіях, особливо з точки зору масштабування та ефективності. Цей процес включає розбиття всієї блокчейн-мережі на менші частини, відомі як "шарди" або сегменти. Кожен такий сегмент або шард обробляє лише певну частину всіх транзакцій, що відбуваються в мережі. Такий підхід дозволяє знизити загальне навантаження на кожен окремий вузол мережі, оскільки замість обробки всіх транзакцій мережі кожен вузол обробляє лише транзакції, що відбуваються в його власному шарді.

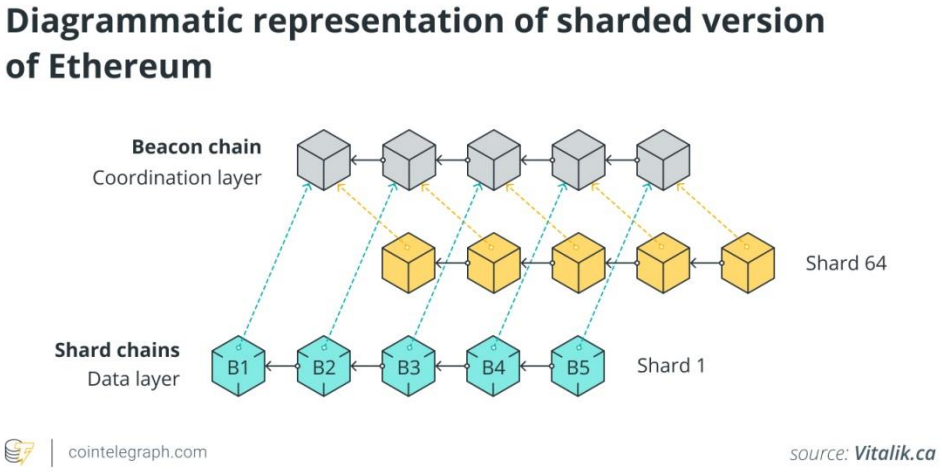



Рисунок 1.11 – Репрезентація роботи Ethereum з використання шардінгу

Шардінг може істотно підвищити загальну пропускну здатність і швидкість

мережі, оскільки транзакції розподіляються між різними сегментами. Це означає, що замість того, щоб кожен вузол мережі виконував одну й ту ж роботу (що є характерним для традиційних блокчейнів), вузли працюють паралельно, значно збільшуючи загальну продуктивність.

Шардінг також має переваги з точки зору масштабованості, оскільки дозволяє мережі обробляти більше транзакцій одночасно, не збільшуючи надмірно вимоги до обчислювальних ресурсів кожного окремого вузла. Це робить блокчейн більш ефективним і доступним для великої кількості користувачів, навіть при зростанні обсягів транзакцій.

Хоча шардінг і вирішує деякі проблеми масштабування, він також вносить певні виклики, зокрема у плані безпеки та узгодження даних між шардами. Розробники та інженери, що працюють над впровадженням шардінгу, повинні враховувати ці аспекти, щоб забезпечити надійну та безпечну роботу розділених мереж..

Layer 2 Рішення:

- Використання додаткових рішень, які працюють поверх основного блокчейну, таких як Lightning Network для Bitcoin або Plasma для Ethereum, є одним із ключових способів збільшення пропускної спроможності мережі. Ці рішення відіграють роль вторинних шарів або "layer 2" рішень, які дозволяють проводити транзакції швидше та ефективніше, не завантажуючи основну блокчейн-мережу.

Lightning Network для Bitcoin є протоколом вторинного шару, який дозволяє користувачам створювати платіжні канали між собою. Це означає, що транзакції можуть відбуватися між учасниками цих каналів миттєво та з мінімальними комісіями, оскільки вони не потребують включення до кожного блоку на основному блокчейні. Закриття каналу і фіналізація всіх транзакцій відбувається лише тоді, коли учасники вирішують завершити взаємодію, і тоді ці дані записуються в основний блокчейн.

З іншого боку, Plasma для Ethereum є концепцією, яка дозволяє створювати "дочірні" блокчейни, пов'язані з основним блокчейном Ethereum. Це дозволяє обробляти транзакції на цих дочірніх блокчейнах, значно зменшуючи навантаження на основну мережу. Кожен дочірній блокчейн може мати свої власні правила і параметри, що забезпечує гнучкість та можливість оптимізації під

конкретні потреби.

Обидва ці рішення вносять великий вклад у вирішення проблеми масштабування блокчейн-мереж. Вони забезпечують можливість обробки великої кількості транзакцій без значного збільшення вартості та часу обробки, що є критично важливим для широкомасштабного прийняття та використання блокчейн-технологій.

Оптимізація Смарт-Контрактів:

- Ретельна оптимізація коду смарт-контрактів може допомогти зменшити навантаження на мережу та збільшити швидкість обробки транзакцій.

Обмеження Блокчейн-Платформ:

- Більшість блокчейн-платформ, включаючи Ethereum, мають обмеження у швидкості обробки транзакцій через механізми консенсусу, такі як PoW або PoS, і обмежену пропускну спроможність блоку.

Транзакційна Вартість та Затримки:

- Зі збільшенням навантаження на мережу зростає вартість транзакцій (газ у Ethereum), а також затримки в обробці транзакцій, що може стати перешкодою для масового прийняття DApps.

Технічні Аспекти Масштабованості

Архітектура Блокчейну:

Архітектура блокчейну є фундаментальним аспектом, який визначає її ефективність, безпеку та масштабованість. Типова архітектура блокчейну передбачає, що кожен вузол в мережі зберігає повну копію ланцюга блоків. Це означає, що кожен вузол має повністю відтворити всі транзакції, які коли-небудь відбувалися в мережі. Цей підхід має кілька важливих переваг, включаючи високий рівень безпеки та децентралізацію, оскільки кожен вузол має однаковий запис усіх транзакцій.

Проте, цей підхід може стати немасштабованим при збільшенні розміру мережі та обсягу даних. Коли кількість транзакцій у мережі зростає, кожен вузол має обробляти і зберігати все більшу кількість даних. Це може створити проблеми, зокрема:

Потреба у значних обчислювальних ресурсах: Для зберігання повної копії блокчейну потрібен значний об'єм пам'яті, а також обчислювальна потужність для

обробки всіх транзакцій.

Збільшення часу синхронізації: Новим вузлам може знадобитися значно більше часу для синхронізації з мережею, оскільки вони повинні завантажити та обробити всю історію транзакцій.

Мережеві обмеження: Збільшення обсягу транзакцій може призвести до перевантаження мережі, особливо якщо швидкість передачі даних є обмеженою.

В результаті, традиційна архітектура блокчейну, хоча і надзвичайно безпечна та децентралізована, може стикатися з викликами масштабування, які потребують розгляду альтернативних підходів або доповнювальних рішень, таких як шардінг, використання легких вузлів або вторинні шари мережі, для підтримки її зростання та розширення функціональності.

Шардінг:

- Шардінг - це процес розділення ланцюга блоків на дрібніші сегменти (шарди), кожен з яких обробляє частину транзакцій. Це може підвищити загальну продуктивність системи.

Off-Chain Рішення:

- Off-Chain рішення, такі як станові канали (state channels) або плазма-ланцюги (plasma chains), є важливими інноваціями в контексті масштабування блокчейн-технологій. Ці рішення дозволяють проводити транзакції поза основним блокчейном, зменшуючи тим самим навантаження на мережу.

Станові Канали (State Channels)

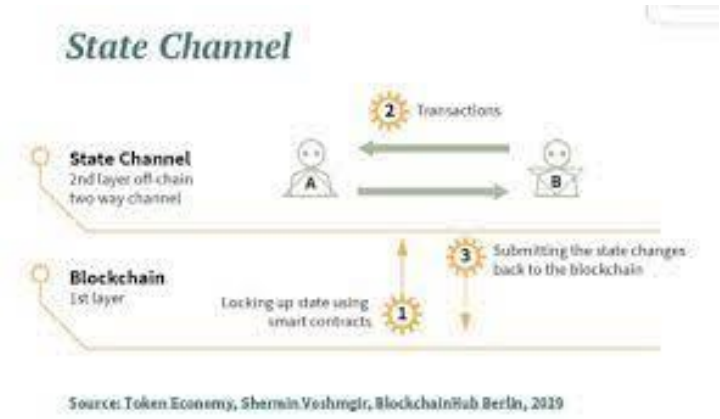



Рисунок 1.12 – Принцип роботи каналів стану

Канали стану – це технологія, що дозволяє користувачам взаємодіяти один з одним безпосередньо, виконуючи транзакції в реальному часі і записуючи лише

кінцевий стан цих транзакцій на блокчейн. Такий підхід значно знижує потребу у включенні кожної транзакції в блокчейн, оскільки більшість транзакцій відбувається "off-chain" та записується на блокчейн лише після закриття каналу. Це дозволяє зменшити час та вартість транзакцій, а також розвантажити мережу від непотрібного навантаження.

Плазма-Ланцюги (Plasma Chains)

Плазма-ланцюги – це концепція, запропонована розробниками Ethereum, яка передбачає створення "дочірніх" блокчейнів, пов'язаних з основним ланцюгом Ethereum. Ці дочірні блокчейни можуть обробляти транзакції та взаємодії окремо від основного ланцюга, зберігаючи при цьому зв'язок з основним блокчейном для забезпечення безпеки та консенсусу. Плазма-ланцюги дозволяють зменшити обсяги оброблюваних даних на основному ланцюгу, що покращує загальну пропускну спроможність та ефективність мережі.

Обидва ці Off-Chain рішення відіграють важливу роль у вирішенні проблеми масштабування блокчейн-мереж. Вони дозволяють збільшити кількість транзакцій, які може обробляти мережа, не збільшуючи при цьому розмірів блоків або частоти їх створення, тим самим зменшуючи навантаження на мережу та забезпечуючи швидшу та ефективнішу обробку транзакцій.

Вплив на Розробку DApps

Проектування для Масштабованості:

- Розробники DApps повинні враховувати обмеження масштабованості при проектуванні своїх додатків, оптимізуючи використання ресурсів та інтегруючи off-chain рішення.

Вибір Платформи:

- Рішення щодо вибору блокчейн-платформи для DApps має враховувати потенційну масштабованість та обмеження відповідної платформи.

Майбутнє Масштабованості

Інновації у Блокчейні

- Розвиток нових технологій та протоколів, наприклад Ethereum 2.0 з його переходом на PoS і шардінг, має на меті вирішити проблеми масштабованості.

Адаптація Ринку:

- З покращенням масштабованості блокчейн-платформ зростає потенціал

для ширшого прийняття та розвитку DApps у різних секторах.

Користувацький Інтерфейс та Досвід:

- DApps часто стикаються з викликами у створенні інтуїтивно зрозумілих та привабливих користувацьких інтерфейсів, що може перешкоджати широкому прийняттю.

Взаємодія з Традиційними Системами:

Інтеграція DApps з Традиційними Системами та Послугами

Інтеграція децентралізованих додатків (DApps) з традиційними системами та послугами представляє собою комплексне завдання, що включає ряд технічних та організаційних викликів.

Основні Виклики Інтеграції

Різниця в Архітектурі:

Централізовані Системи: Детальний Огляд

Основи Централізованих Систем

Означення:

- Централізовані системи - це архітектурні моделі, де обробка даних, управління ресурсами та взаємодія з користувачами зосереджені на одному або кількох центральних серверах. Вони контролюють всю діяльність та ресурси системи.

Архітектура:

- Типова централізована система включає основний сервер (або кластер серверів), який забезпечує обробку даних, зберігання, управління транзакціями, а також веб-сервіси.

Компоненти та Їх Роль

. Центральний Сервер:

Детальний Опис Центрального Сервера у Централізованих Системах

Ядро Централізованої Системи

. Функціональність Центрального Сервера:

- Обробка Запитів: Центральний сервер приймає, обробляє та відповідає на запити від клієнтських додатків. Це може включати запити на дані, транзакції або будь-які інші взаємодії з системою.

- Управління Базою Даних: Сервер управляє централізованою базою даних,

здійснюючи операції зберігання, оновлення, видалення та витягування даних.

- Виконання Бізнес-Логіки: Всі бізнес-правила та логіка обробки даних зосереджені на сервері. Це включає обробку транзакцій, управління користувацькими акаунтами, валідацію даних тощо.

Архітектура:

- Фізичний vs Віртуальний Сервер:

- Фізичний сервер відноситься до конкретного фізичного обладнання, розміщеного в дата-центрі.

- Віртуальний сервер (або хмарний сервер) функціонує у середовищі хмарних технологій, де ресурси можуть бути легко масштабовані та адаптовані до потреб системи.

. Системне Отчення:

- Центральний сервер забезпечує системне оточення для веб-додатків, включаючи веб-сервери, додаткові служби, такі як кешування, балансування навантаження, і інструменти моніторингу.

Важливість Центрального Сервера

Центральний сервер у централізованих системах відіграє ключову роль у обробці та управлінні запитами та даними. Він вимагає ретельного управління, оскільки його продуктивність, надійність та безпека безпосередньо впливають на функціонування всієї системи. Розуміння та ефективне управління центральним сервером є критично важливими для успішної експлуатації централізованих систем.

Надійність і Продуктивність:

- Висока продуктивність і надійність центрального сервера критично важливі для ефективної роботи централізованої системи. Неполадки або збої сервера можуть призвести до відмови всієї системи.

Безпека:

- Центральний сервер має забезпечувати високий рівень безпеки, оскільки він зберігає важливі дані та управляє критичними для бізнесу процесами. Це включає захист від зовнішніх атак, забезпечення конфіденційності даних та інтегритету інформації.

Виклики у Роботі з Центральним Сервером

Масштабування:

- Один з основних викликів централізованих систем полягає у масштабуванні. Розширення потужності фізичного сервера може бути дорогим та часомістким, а в хмарних середовищах може вимагати додаткового планування та ресурсів.

Резервування та Відновлення:

- Важливо мати ефективні системи резервного копіювання та відновлення для центрального сервера, щоб забезпечити безперебійну роботу системи та захист даних у випадку збоїв або катастроф.

Клієнтська Частина:

- Користувачі взаємодіють з системою через клієнтські додатки (наприклад, веб-браузери), які відправляють запити на сервер та отримують відповіді.

База Даних:

- Централізована система зазвичай містить центральну базу даних, яка зберігає всю необхідну інформацію та забезпечує її доступність для сервера.

Переваги та Недоліки

Переваги:

- Простота Управління: Централізовані системи легші в управлінні та моніторингу, оскільки всі ресурси зосереджені в одному місці.

- Ефективність Обробки Даних: Завдяки концентрації обчислювальних потужностей, централізовані системи можуть бути ефективні у великих обсягах даних.

Недоліки:

- Єдина Точка Відмови: Централізовані системи вразливі до відмов, оскільки збій одного сервера може вивести з ладу всю систему.

- Масштабованість: Розширення централізованих систем може бути складним і дорогим, особливо при значному збільшенні навантаження.

Технічні Аспекти

Безпека:

- Централізовані системи вимагають ретельного управління безпекою, оскільки вони можуть бути мішенню для хакерських атак.

. Обслуговування та Оновлення:

- Регулярне обслуговування та оновлення необхідні для забезпечення стабільної роботи та безпеки системи.

Централізовані системи досі є основою багатьох веб-додатків та сервісів, незважаючи на появу децентралізованих технологій. Вони пропонують стабільність, ефективність та простоту управління, але також мають свої обмеження, особливо у випадках, де висока надійність та безперервна доступність є критично важливими.

- Децентралізовані Системи: DApps працюють на блокчейн-платформах, де дані зберігаються та обробляються у розподіленому реєстрі, що вимагає іншого підходу до обробки запитів та управління даними.

Різниця у Стандартах:

- Інтеграція DApps з традиційними системами може стикатися з відмінностями у протоколах, форматах даних та інтерфейсах програмування додатків (API), що ускладнює взаємодію між різними системами.

. Безпека та Довіра:

Безпека та Довіра при Інтеграції DApps з Централізованими Системами

Основні Аспекти Безпеки

1. Захист Даних:

- Забезпечення безпеки даних є важливим при інтеграції DApps з централізованими системами, особливо коли в обігу чутливі або особисті дані. Це означає використання сильних криптографічних алгоритмів для шифрування даних під час передачі та зберігання.

2. Аутентифікація та Авторизація:

- Належні процедури аутентифікації та авторизації є ключовими для забезпечення того, що тільки уповноважені користувачі мають доступ до даних та функцій DApps. Це може включати використання многофакторної аутентифікації та управління доступом на основі ролей.

3. Безпека API:

- При інтеграції DApps з централізованими системами важливо забезпечити безпеку API, які використовуються для взаємодії. Це включає в себе захист від загроз, таких як SQL-ін'єкції, cross-site scripting (XSS) та інших видів атак.

4. Захист Від DDoS Атак:

- DApps та централізовані системи можуть бути вразливими до розподілених атак типу відмови в обслуговуванні (DDoS). Застосування заходів захисту, як-от географічне розподілення ресурсів та фільтрація трафіку, є важливим.

Аспекти Довіри

1. Прозорість Операцій:

- Для забезпечення довіри важливо зробити операції та процеси, які відбуваються у DApps, прозорими для користувачів. Це може означати надання зрозумілих логів транзакцій або аудитів системи.

2. Сертифікація та Відповідність Стандартам:

- Дотримання визнаних стандартів безпеки та отримання відповідних сертифікацій може збільшити довіру користувачів та партнерів.

3. Політика Конфіденційності та Використання Даних:

- Ясне та прозоре інформування користувачів про політику конфіденційності та використання зібраних даних підвищує довіру та забезпечує відповідність законодавчим нормам.

Технічні Рішення

1. Системи Керування Ідентичністю:

- Впровадження надійних систем управління ідентичністю та контролю доступу є важливим для забезпечення безпечної взаємодії між DApps та централізованими системами.

2. Шифрування Каналів Зв'язку:

- Використання TLS/SSL для шифрування даних, переданих між DApps та централізованими серверами, забезпечує безпеку даних під час транзакцій.

3. Резервне Копіювання та Відновлення:

- Реалізація надійних процесів резервного копіювання та відновлення допомагає забезпечити безперебійність бізнес-процесів та захист даних.

Інтеграція DApps з централізованими системами вимагає комплексного підходу до безпеки та довіри. Важливо враховувати не тільки технічні аспекти, але й питання конфіденційності, дотримання стандартів, прозорість процесів, а також розвиток відносин довіри з користувачами та партнерами. Ефективне управління

цими аспектами може значно покращити успішність та прийняття DApps у широкому колі користувачів.

Технічні Аспекти Інтеграції

1. Інтерфейси для Взаємодії:

- Розробка спеціалізованих API або використання існуючих інструментів для забезпечення взаємодії між DApps та традиційними системами. Це може включати створення шлюзів або адаптерів, які перетворюють запити і відповіді між різними форматами.

2. Синхронізація Даних:

- Налагодження процесів синхронізації даних між блокчейнами та традиційними базами даних, забезпечуючи консистентність та актуальність інформації.

3. Аутентифікація та Доступ:

- Реалізація надійних механізмів аутентифікації та управління доступом для забезпечення безпечного взаємодії між DApps та традиційними системами.

Організаційні Аспекти

1. Порозуміння Між Різними Стейкхолдерами:

- Важливість забезпечення чіткого порозуміння та взаємодії між різними учасниками процесу інтеграції, включаючи розробників DApps, адміністраторів традиційних систем, та кінцевих користувачів.

2. Навчання та Підтримка:

Процес Інтеграції та Необхідність Навчання та Підтримки

Інтеграція децентралізованих додатків (DApps) з традиційними системами вимагає глибокого розуміння різних технологій та методологій. Цей процес часто включає в себе спеціальне навчання та підтримку для забезпечення ефективної взаємодії між системами.

Освітні Потреби

1. Технічне Навчання:

- Розуміння Блокчейн-технологій: Навчання розробників основам блокчейну, включаючи принципи роботи смарт-контрактів, криптографію, консенсусні алгоритми.

- Технології Інтеграції: Освоєння інструментів і методів інтеграції, таких як API, middleware, протоколи обміну даними.

2. Розробка Інтерфейсів:

- Навчання у створенні користувацьких інтерфейсів, які ефективно взаємодіють як з блокчейн-мережами, так і з традиційними системами.

3. Безпека та Конфіденційність:

- Освіта з питань безпеки, шифрування, відповідності законодавчим стандартам та захисту даних.

Підтримка

1. Технічна Підтримка:

- Надання технічної підтримки для розробників та адміністраторів системи для вирішення поточних проблем, пов'язаних з інтеграцією.

2. Документація та Ресурси:

- Створення та утримання детальної документації, яка охоплює архітектурні стандарти, використання API, кращі практики безпеки та рекомендації з розробки.

3. Форуми та Спільноти:

- Створення та підтримка спільнот, де розробники можуть ділитися досвідом, вирішувати проблеми та обмінюватися ідеями.

Впровадження та Тестування

1. Пілотні Проекти:

- Реалізація пілотних проектів перед повномасштабною інтеграцією для оцінки ефективності взаємодії між системами.

2. Тестування Сумісності:

- Проведення ретельного тестування для перевірки сумісності, продуктивності та безпеки інтегрованих систем.

Організаційний Розвиток

1. Управління Змінами:

- Розробка стратегій управління змінами для забезпечення плавної адаптації персоналу та процесів до нової інтегрованої системи.

2. Неперервне Навчання:

- Впровадження програм неперервного навчання та розвитку для підтримки актуальності навичок та знань у швидко змінюваному технологічному середовищі.

Процес інтеграції DApps з традиційними системами - це складне завдання, яке вимагає не тільки технічних навичок, але й організаційного розвитку, навчання, підтримки та управління змінами. Ефективне впровадження цих аспектів є ключем до успішної інтеграції та використання потенціалу децентралізованих технологій у традиційних веб-системах.

Інтеграція DApps з традиційними системами та послугами представляє собою складний процес, що вимагає глибокого технічного розуміння та врахування організаційних аспектів. Ефективна інтеграція може відкрити нові можливості для використання блокчейн-технологій у різних сферах, але вимагає ретельного планування, тестування та управління проектом.

Дослідження прикладів децентралізованих веб-додатків виявляє як потенційні переваги, так і виклики, з якими стикаються розробники та користувачі. Ці приклади демонструють різноманітність застосування блокчейн-технологій в різних сферах та підкреслюють важливість продовження інновацій та досліджень у цій області.

# 2 АНАЛІЗ І ПОРІВНЯННЯ ДЕЦЕНТРАЛІЗОВАНИХ ТЕХНОЛОГІЙ

## 2.1 Огляд Різних Децентралізованих Мереж та Платформ

Ethereum

Як лідер у створенні децентралізованих додатків (DApps), Ethereum пропонує можливості для створення смарт-контрактів та підтримує широкий спектр застосунків, від DeFi до ігор та соціальних мереж.

Bitcoin

Bitcoin, як перша криптовалюта, відіграє важливу роль у світі цифрових валют і блокчейн технологій. Вона була створена в 2009 році невідомою особою або групою осіб під псевдонімом Сатоші Накамото. Основною ідеєю Bitcoin є створення децентралізованої валюти, яка не підпорядковується жодній центральній владі чи уряду.

Важливою особливістю Bitcoin є те, що вона не призначена для розробки децентралізованих додатків (DApps). На відміну від Ethereum та інших блокчейн-платформ, які були створені з урахуванням можливості розробки та запуску DApps, Bitcoin зосереджений переважно на функції валюти.

Оглядаючи різноманітність децентралізованих мереж та платформ, можна виділити кілька ключових аспектів, на яких вони базуються, взявши за основу ідеї та концепції, започатковані Bitcoin.

1. Децентралізація: Ця концепція є краєвугільним каменем для багатьох криптовалют та блокчейн-платформ. Децентралізація відсилає до відсутності центральної контрольної влади в мережі, що дозволяє кожному учаснику мережі мати рівні права та обов'язки. Це створює більш стійку та безпечну систему, оскільки відсутність централізованого контролю знижує ризики цензури та зловмисних атак.

2. Блокчейн як основа безпеки та прозорості: Блокчейн-технології дозволяють записувати транзакції у вигляді ланцюга блоків, де кожен наступний блок містить хеш попереднього, створюючи незмінну структуру даних. Це забезпечує високий рівень безпеки та прозорості, оскільки будь-яка спроба змінити

інформацію в одному блоку вимагатиме зміни всіх наступних блоків, що є практично неможливим без виявлення.

3. Розширення функціональності через смарт-контракти та DApps: Платформи, такі як Ethereum, йдуть далі, ніж Bitcoin, включаючи в себе можливості смарт-контрактів та створення децентралізованих додатків (DApps). Смарт-контракти - це програми, які автоматично виконують умови договору, коли задовольняються певні критерії, що значно розширює можливості використання блокчейну, від автоматизації фінансових операцій до створення складних децентралізованих систем.

4. Інновації та різноманітність у використанні блокчейну: Численні інші блокчейн-платформи продовжують розвивати ці ідеї, пропонуючи унікальні рішення та застосування. Наприклад, деякі зосереджуються на підвищенні конфіденційності, інші на покращенні масштабування або створенні спеціалізованих функцій для конкретних галузей.

Subsocial

Subsocial є прикладом інноваційної децентралізованої мережі та платформи, яка розроблена з урахуванням сучасних тенденцій у сфері блокчейн-технологій та децентралізованих додатків (DApps). Ця платформа зосереджена на створенні соціальної мережі, яка працює на блокчейні, пропонуючи унікальні особливості та можливості для користувачів і розробників. Давайте розглянемо основні аспекти Subsocial:

1. Блокчейн та Децентралізація: Subsocial базується на блокчейні, забезпечуючи високий рівень децентралізації. Це означає, що дані користувачів та їх взаємодії зберігаються у відкритому і незмінному ланцюзі блоків, що гарантує безпеку, прозорість та незалежність від централізованих організацій.

2. Соціальна Мережа на Блокчейні: Subsocial представляє собою унікальне та новаторське злиття концепцій соціальних мереж з передовими блокчейн-технологіями, стаючи однією з перших платформ, яка реалізує це поєднання. В основі цієї платформи лежить ідея створення соціальної мережі, яка функціонує на блокчейні, що приносить ряд переваг, які традиційні соціальні мережі просто не можуть запропонувати.

Перш за все, Subsocial дозволяє користувачам створювати контент. Це

можуть бути пости, блоги, відео, зображення та інші типи медіаконтенту. Важливо, що вся ця інформація записується на блокчейн, що забезпечує її незмінність та вічну доступність. Таким чином, користувачі можуть бути впевнені, що їхній контент залишиться недоторканим та не зможе бути видаленим або зміненим без їх відома.

Крім того, Subsocial надає можливість взаємодії між користувачами. Це не просто коментування постів чи обмін повідомленнями; це створення справжніх зв'язків та взаємодій у децентралізованому середовищі. Користувачі можуть встановлювати взаємозв'язки, слідкувати одне за одним та взаємодіяти в контексті, який вони вважають за потрібне.

Однією з найцікавіших особливостей Subsocial є здатність користувачів створювати групи та спільноти. Ці спільноти можуть бути орієнтовані на певні інтереси, хобі, професійні сфери або будь-які інші аспекти. Така організація дозволяє учасникам знаходити однодумців та обмінюватися ідеями та досвідом у захищеному та відкритому середовищі.

На додаток до цього, блокчейн-технологія, яка лежить в основі Subsocial, забезпечує безпеку та прозорість у всіх цих процесах. Кожна транзакція, кожен обмін інформацією або створення контенту відбувається з використанням блокчейну, гарантуючи, що всі взаємодії є відкритими та не можуть бути маніпульовані ззовні. Це створює середовище, де користувачі можуть вільно висловлюватися та діяти, знаючи, що їхні права та дані захищені.

В цілому, Subsocial відкриває нові горизонти для соціальних мереж, впроваджуючи блокчейн для створення більш безпечного, прозорого та користувацьки орієнтованого досвіду. Воно представляє собою зразок того, як технологія може бути використана для підвищення стандартів взаємодії та спілкування в онлайн-просторі.

3. Токенізація та Економічні Стимули: Subsocial використовує токени для стимулювання активності в мережі. Користувачі можуть отримувати токени за створення популярного контенту, модерацію спільнот або інші вклади в екосистему. Це створює економічну систему, де учасники мотивовані підтримувати та розвивати мережу.

4. Можливості для Розробників: Subsocial надає розробникам інструменти та

API для створення власних децентралізованих додатків, які інтегруються з соціальною мережею. Це відкриває двері для широкого спектру додатків, від простих блогів до складних соціальних платформ.

5. Комплексність і Масштабованість: Як і більшість децентралізованих платформ, Subsocial стикається з викликами, пов'язаними з масштабованістю та комплексністю обробки великих об'ємів даних та транзакцій. Розвиток технологій, таких як sharding або off-chain рішення, може допомогти вирішити ці проблеми.

Загалом, Subsocial є важливим внеском у світ блокчейн-технологій і децентралізованих мереж, пропонуючи новаторський підхід до соціальних мереж і створення контенту. Її розвиток і адаптація користувачами буде визначальним для майбутнього цього напрямку в технологіях.

Cosmos (ATOM):

Cosmos є однією з провідних платформ у сфері блокчейн, яка вирішує дві основні проблеми, з якими стикаються багато блокчейн-мереж: масштабованість і міжопераційність. Її унікальна архітектура та технологічні інновації дозволяють різним блокчейнам не тільки ефективно взаємодіяти один з одним, але й масштабуватися відповідно до своїх потреб. Ось детальний огляд ключових аспектів технології та архітектури Cosmos:

Тендермінт (Tendermint)

Tendermint Core є основою Cosmos, що виступає як консенсусний механізм і мережевий протокол. Це BFT (Byzantine Fault Tolerant) консенсусний механізм, що означає, що він може працювати навіть якщо частина вузлів мережі діє зловмисно або є недоступною. Tendermint дозволяє розробникам побудовувати блокчейни без потреби створювати власний механізм консенсусу, значно спрощуючи процес розробки.

Cosmos SDK

Cosmos SDK є фреймворком для створення блокчейн-додатків. Він дозволяє розробникам створювати блокчейни з використанням модульної архітектури, де різні компоненти (такі як системи управління аккаунтами, IBC (Inter-Blockchain Communication), управління токенами) можуть бути легко інтегровані. Це дозволяє створювати масштабовані та гнучкі блокчейн-рішення.

Inter-Blockchain Communication (IBC)

IBC є ключовою інновацією Cosmos, яка дозволяє безпечну та ефективну міжопераційність між різними блокчейнами. Це протокол для надійного та безпечного обміну даними та вартостями між різними блокчейнами, які підтримують цей протокол. IBC розширює можливості блокчейнів, дозволяючи їм взаємодіяти в багатоланцюжковому середовищі.

Космос-Хаб (Cosmos Hub)

Cosmos Hub є центральним блокчейном у екосистемі Cosmos, який використовується для з'єднання різних блокчейнів через IBC. Це дозволяє блокчейнам, підключеним до Cosmos Hub, безпечно обмінюватися даними та вартостями, створюючи екосистему взаємопідключених блокчейнів.

Масштабованість

Cosmos вирішує проблему масштабованості, дозволяючи кожному блокчейну в екосистемі оброб

Polkadot:

- Подібно до Cosmos, Polkadot забезпечує міжблокчейнну взаємодію, дозволяючи різним блокчейнам передавати повідомлення та вартості в єдиній децентралізованій системі [11].

Загалом, кожна з цих платформ вносить свій вклад у розвиток екосистеми блокчейну, підтримуючи інновації та розширюючи можливості використання цих технологій в різних сферах діяльності.

IPFS

10 жовтня 2018 року було представлено новий протокол для створення децентралізованої файлообмінної мережі, який отримав назву IPFS. Цей протокол використовує ключові ідеї BitTorrent та доповнює їх новими вдосконаленнями, що дозволяють використовувати IPFS як платформу для різноманітних децентралізованих застосунків. У цьому розділі ми розглянемо основні поняття IPFS, переваги протоколу порівняно з іншими методами створення децентралізованих файлообмінних мереж, а також способи застосування та потенційне використання IPFS у реалізації децентралізованих застосунків.

Суть роботи цієї системи полягає в тому, що користувачі не завантажують файли з централізованих серверів, а безпосередньо обмінюються ними між собою. Концепція IPFS досить схожа на концепцію Всесвітньої мережі (World Wide Web).

Кожному файлу в системі присвоюється унікальний ідентифікатор, який є його хеш-значенням. Для відстеження історії кожного файлу використовується Git (система управління версіями). Такий підхід дозволяє забезпечити доступ до найактуальнішої версії контенту та гарантувати його автентичність. Пошук конкретного файлу можливий як за його ідентифікатором, так і за зрозумілими для людини назвами. Для цього використовується децентралізована система імен IPNS.

Основні принципи протоколу:

IPFS (InterPlanetary File System) — це протокол для децентралізованої файлообмінної системи. Протокол використовує концепцію DHT (розподіленої хеш-таблиці) для розподілу та пошуку контенту між учасниками, а також MDAG (напрямлений ациклічний граф Меркла) для створення зручної структури зв'язків контенту [5]. Як і BitTorrent, IPFS не передбачає зберігання всіх файлів на кожному з вузлів системи: кожен учасник зберігає лише ту частину контенту, яку вважає необхідною. IPFS (InterPlanetary File System) є інноваційним протоколом, створеним для децентралізованого обміну файлами. Основні особливості та принципи роботи IPFS включають наступні аспекти:

- Використання DHT (розподіленої хеш-таблиці): DHT, або розподілена хеш-таблиця, є ключовим елементом у структурі IPFS. Це спосіб організації даних, який дозволяє ефективно розподіляти та знаходити контент у децентралізованій мережі. Кожен файл або фрагмент даних у системі IPFS має унікальне хеш-значення, що використовується для його ідентифікації та пошуку в мережі. Це дозволяє учасникам мережі знаходити потрібні файли без необхідності звертатися до централізованого сервера.

- Використання MDAG (напрямлений ациклічний граф Меркла): MDAG, або напрямлений ациклічний граф Меркла, використовується в IPFS для створення структури зв'язків між контентом. Це дозволяє ефективно організувати та лінкувати різні частини даних, спрощуючи доступ та управління контентом в мережі. Кожен блок даних містить інформацію не тільки про самі дані, але й про їх зв'язки з іншими блоками, створюючи деревоподібну структуру.

- Розподілене зберігання контенту: Відмінною рисою IPFS є те, що

він не вимагає зберігання всіх файлів на кожному вузлі системи. Натомість, файли розподіляються між учасниками мережі, де кожен учасник зберігає тільки ту частину контенту, яка йому необхідна або якою він зацікавлений. Це підвищує ефективність зберігання та доступу до даних, а також робить систему більш стійкою до збоїв і цензури.

- Деревоподібна структура зв'язків контенту: Схоже на традиційну файлову систему, IPFS використовує деревоподібну структуру для організації контенту, що полегшує навігацію та взаємодію з даними.

- Перевірка та видалення дублікатів файлів: IPFS автоматично виявляє та усуває дублікати файлів у мережі, забезпечуючи оптимізацію використання простору та ресурсів.

- Можливість версіонування історій файлів: IPFS підтримує версіонування файлів, що дозволяє зберігати історію змін та версій різних файлів, полегшуючи відновлення попередніх версій та відстеження змін у документах.

Ці особливості роблять IPFS потужною та гнучкою системою для децентралізованого зберігання та обміну даними, відкриваючи нові можливості для створення і використання децентралізованих застосунків.

Кожен вузол в IPFS зберігає власну таблицю хеш-значень, яка містить зв'язок між конкретним ідентифікатором та даними, які можуть бути або частиною контенту, або мережевою адресою іншого учасника мережі. Особливістю IPFS є підтримка версіонування контенту, що дозволяє при оновленні документа зв'язати нову версію зі старою, схоже на роботу Google Docs або GitHub.

Як працює IPFS:

IPFS працює подібно до BitTorrent, використовуючи модель адресації контенту замість джерел контенту. Протокол передбачає розділення контенту на файли (фрагменти) до 256 КБ кожен, кожному з яких присвоюється унікальний хеш-ідентифікатор. Протокол не обмежує вибір хеш-функції, дозволяючи використовувати різні алгоритми.

Звісно, ось більш детальний опис чотирьох основних типів файлів у IPFS:

1. Blob (набір даних):

- Blob у IPFS відноситься до базового типу файлу, який представляє собою набір двійкових даних або вмісту. Це може бути будь-який тип даних, наприклад, текстовий документ, зображення, аудіофайл або відео. Blob не містить метаданих або вказівок на інші файли чи об'єкти. Це найпростіший тип контенту, який зберігається в IPFS.

2. List (список):

- List у IPFS — це структура даних, яка використовується для представлення послідовності об'єктів, зазвичай Blob або інших List. Це може бути порівняно з папкою в традиційній файловій системі, яка містить список файлів або підпапок. Він дозволяє організувати та групувати пов'язані об'єкти або файли, спрощуючи їх керування та доступ.

3. Commit (стан):

- "Commit" у IPFS використовується для представлення стану або версії певного об'єкта чи групи об'єктів. Це схоже на коміт у системах контролю версій, наприклад, Git. Він фіксує певний стан файлу або набору файлів у певний момент часу. Commit може містити метадані про зміни, які були здійснені, автора змін, час здійснення змін тощо. Це дозволяє відстежувати історію змін і відновлювати попередні стани файлів.

4. Tree (дерево):

- Tree у IPFS — це структура, яка представляє ієрархічну організацію файлів та папок.

- Вона схожа на структуру директорій в традиційних файлових системах, дозволяючи влаштовувати файли в папках та підпапках.

- Tree об'єкт дозволяє ефективно структурувати та організовувати велику кількість файлів і папок, забезпечуючи зручний доступ і навігацію по файловій системі IPFS.

Кожен з цих типів файлів відіграє важливу роль у забезпеченні функціональності та гнучкості IPFS, дозволяючи ефективно управляти та організовувати різноманітний контент у децентралізованому середовищі.

## 2.2 **Порівняння SDK для розробки децентралізованих систем**

1. Truffle Suite (Ethereum):

- Набір інструментів для розробки Ethereum DApps, включаючи тестування, розгортання смарт-контрактів та управління розробкою.

2. Hyperledger Composer:

Hyperledger Composer є передовим інструментом, призначеним для розробки блокчейн-додатків на базі Hyperledger Fabric. Цей інструмент розроблений з метою спрощення процесу розробки блокчейн-рішень, роблячи його більш доступним та менш трудомістким для розробників.

Основна ідея Hyperledger Composer полягає у наданні користувачам зрозумілого та зручного інтерфейсу для створення та управління блокчейн-мережами. Це включає в себе визначення активів, учасників, правил доступу, а також логіки транзакцій, що дозволяє розробникам створювати складні додатки з меншими зусиллями порівняно з безпосередньою роботою на рівні блокчейну.

Ключовим аспектом Hyperledger Composer є його модульна архітектура, яка дозволяє легко інтегрувати розроблені додатки з різними системами та сервісами. Така структура сприяє швидкому прототипуванню та тестуванню блокчейн-додатків, що є важливим в аспектах швидкісного розвитку ринку та постійно змінюваних технологічних вимог.

Hyperledger Composer також надає засоби для візуалізації блокчейн-мереж, що полегшує розуміння та аналіз структури та динаміки мережі. Це особливо корисно для комплексного розуміння взаємодії різних компонентів системи та їх впливу на загальну ефективність та безпеку додатку. Завдяки своїй гнучкості, масштабованості та користувацько-орієнтованому підходу, Hyperledger Composer відкриває нові можливості для підприємств та індивідуальних розробників у світі блокчейн-технологій, дозволяючи їм ефективно реалізовувати інноваційні рішення на основі блокчейну.

3. Cosmos SDK:

Cosmos SDK є високопродуктивною рамкою, розробленою для створення блокчейн-додатків з підвищеною міжблокчейнною міжопераційністю та масштабованістю. Цей інструмент зосереджений на спрощенні процесу розробки,

надаючи розробникам набір потужних інструментів та компонентів, які полегшують створення ефективних і гнучких блокчейн-рішень [7]. Cosmos SDK забезпечує високу ступінь міжопераційності між різними блокчейнами, дозволяючи легко інтегрувати та взаємодіяти з різними мережами і протоколами. Це особливо важливо у світі, де існує безліч блокчейн-платформ, і потреба у масштабуванні та ефективному обміні даними між ними стає все більш актуальною. Таким чином, Cosmos SDK відіграє ключову роль у створенні інноваційних блокчейн-додатків, здатних ефективно функціонувати в широкому та різноманітному екосистемному середовищі.

4. Subsocial SDK

Subsocial SDK (Software Development Kit) є частиною Subsocial - відкритої платформи, яка дозволяє створювати децентралізовані соціальні мережі та маркетплейси на основі блокчейн технологій. Subsocial SDK надає розробникам набір інструментів, бібліотек та API для легкої інтеграції функціональності Subsocial в їх додатки або для створення нових застосунків на базі Subsocial. SDK базується на блокчейні і інтегрується з IPFS (InterPlanetary File System) для зберігання великих даних, таких як зображення, відео та інші медіафайли. Це дозволяє створювати децентралізовані соціальні мережі з надійним зберіганням великих обсягів контенту [9].

SDK також підтримує розумні контракти, які можна використовувати для автоматизації процесів, вирішення спорів та інших функцій. Воно забезпечує гнучкість управління доступом, дозволяючи налаштовувати приватність, права доступу та інші параметри безпеки. SDK включає API для спрощення інтеграції функціональності Subsocial у сторонні застосунки. API дозволяє взаємодіяти з різними компонентами системи, включаючи зберігання даних, управління аккаунтами, транзакції та багато іншого. SDK надає широкі можливості для кастомізації та адаптації функціональності застосунків під конкретні потреби та вимоги. Важливий акцент робиться на інструментах для створення та управління спільнотами, соціальними взаємодіями, публікаціями постів, коментарями, голосуваннями та іншими.

Для полегшення розробки та інтеграції, Subsocial SDK забезпечує розробників документацією, прикладами коду, інструкціями та іншими ресурсами.

Subsocial SDK відкриває широкі можливості для розробки децентралізованих соціальних мереж, блог-платформ, маркетплейсів та інших застосунків, надаючи розробникам потужні інструменти для роботи з блокчейн-технологіями та децентралізованим зберіганням даних.

### 2.3 Оцінка фронтенд-бібліотек та фреймворків для розробки dapps

1. Основні Характеристики:

- React.js - це декларативна, ефективна та гнучка JavaScript бібліотека для побудови користувацьких інтерфейсів. Особливість React полягає у використанні компонентного підходу, що дозволяє розбити інтерфейс на незалежні, повторно використовувані частини [2].

React.js є вибором, який поєднує в собі швидкість, гнучкість та ефективність, роблячи його ідеальним інструментом для розробки різноманітних DApps. Чи то торгові платформи, ігри або соціальні мережі - React дозволяє створювати динамічні, візуально привабливі інтерфейси, які підвищують залучення користувачів та покращують загальний користувацький досвід. Завдяки своїм унікальним особливостям та широким можливостям застосування, React.js продовжує бути одним з найбільш популярних виборів для розробників DApps.

2. Переваги для DApps:

- Модульність: Легкість управління станом і данними великих додатків завдяки модульній архітектурі.

- Спільнота та Екосистема: Велика спільнота розробників і готових рішень, що полегшують інтеграцію з різними блокчейн-технологіями.

3. Сценарії Використання:

1. React.js

React.js, як фронтенд-бібліотека, створена Facebook, займає провідну позицію у розробці децентралізованих додатків (DApps). Це інструмент, який забезпечує швидку та ефективну взаємодію з користувачами, що є критично важливим у таких сферах, як торгові платформи, ігри на блокчейні та соціальні мережі.

Переваги React.js для Розробки DApps

1. Компонентний Підхід:

- React.js використовує компонентний підхід, що дозволяє розбити інтерфейс додатку на повторно використовувані елементи. Це полегшує процес розробки та тестування, дозволяючи створювати складні інтерфейси з легко управляємими частинами.

2. Декларативний Стиль Програмування:

- Робота з React включає декларативне програмування, що робить код більш читабельним та легшим для зрозуміння. Розробники описують, як має виглядати UI, а React самостійно керує оновленнями DOM при зміні стану додатку.

3. Висока Продуктивність:

- React ефективно оптимізує оновлення DOM, використовуючи віртуальний DOM, що знижує кількість взаємодій з реальним DOM та підвищує продуктивність додатків.

Застосування React.js у Різних Сферах DApps

1. Торгові Платформи на Блокчейні:

- У торгових платформах на блокчейні, де важлива швидка реакція на ринкові зміни та висока продуктивність, React.js дозволяє створювати плавні та відгукові інтерфейси. Його здатність швидко оновлювати візуальні компоненти відповідно до змін у реальному часі є ключовою для торгових додатків.

2. Ігри на Блокчейні:

- У сфері ігор на блокчейні, де потрібна інтерактивність та висока якість візуалізації, React.js сприяє створенню захоплюючих інтерфейсів. Він забезпечує розробників інструментами для створення динамічних ігрових елементів, що реагують на дії користувача в реальному часі.

3. Соціальні Мережі на Блокчейні:

- Для соціальних мереж, створених на основі блокчейну, React.js допомагає у розробці інтерфейсів, які підтримують велику кількість соціальних взаємодій та даних користувачів. Ефективне управління станом та оновлення інтерфейсу забезпечують плавну взаємодію користувачів з платформою.

Vue.js

1. Особливості Vue.js:


Рисунок 2.1 – Логотип Vue.js

- Vue.js - це прогресивний JavaScript фреймворк, який забезпечує реактивну та компонентно-орієнтовану розробку. Його простота та гнучкість роблять його популярним вибором для швидкого розвитку проектів [3] .

2. Переваги для DApps:

- Легкість вивчення та Розвитку: Вважається одним з найбільш простих у вивченні фреймворків, що робить його доступним для нових розробників.

- Гнучкість: Можливість легко інтегруватися з іншими бібліотеками та існуючими проектами.

3. Використання у DApps:

- Vue.js підходить для розробки легких DApps, де потрібна швидка взаємодія з користувачем і простота впровадження.

Angular

1. Характеристики Angular:

Angular є одним із найпопулярніших та найпотужніших фреймворків для розробки веб-додатків. Розроблений командою з Google, Angular вирізняється своїм багатим набором функціональностей, що дозволяють розробникам створювати складні, високопродуктивні та легко підтримувані веб-додатки [4]. Основні можливості Angular включають:

1. Двосторонній Байндінг (Two-Way Data Binding): Ця особливість дозволяє автоматично синхронізувати дані між моделлю та виглядом (view) у веб-додатку. Зміни, внесені у вигляді, негайно відображаються в моделі та навпаки. Це забезпечує більш ефективне управління даними і знижує необхідність в ручному оновленні елементів інтерфейсу.

2. Маршрутизація (Routing): Angular дозволяє розробникам легко налаштовувати навігацію між різними виглядами та компонентами в додатку. Маршрутизація в Angular підтримує складні сценарії навігації та динамічне завантаження компонентів.

3. Модульність (Modularity): Angular організований у модулі, які групують споріднені компоненти, сервіси, директиви, пайпи тощо. Це сприяє кращій організації коду, його повторному використанню та полегшує тестування.

4. Вбудоване Тестування (Built-in Testing): Angular пропонує вбудовані інструменти для тестування, які дозволяють розробникам легко писати та виконувати юніт-тести та інтеграційні тести. Такий підхід забезпечує високу якість коду та додатку в цілому.

Angular широко використовується розробниками завдяки своїй гнучкості, потужності та підтримці від великої спільноти. Цей фреймворк ідеально підходить для розробки складних односторінкових додатків (SPA) та інших великих веб-проектів, які вимагають масштабування, високої продуктивності та легкості в підтримці..

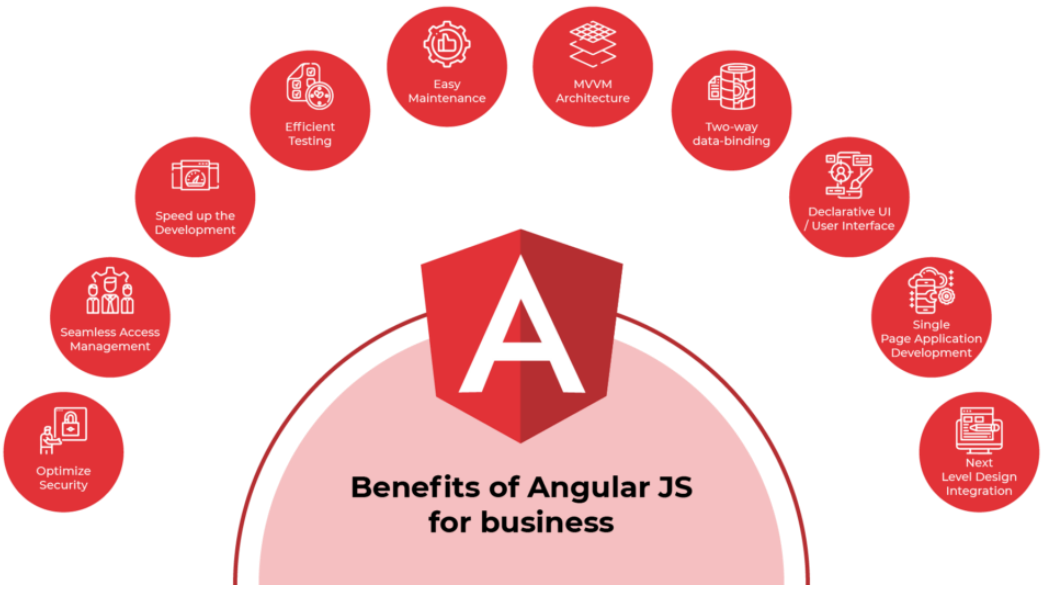


Рисунок 2.2 – Візуалізація переваг Angular

2. Переваги для DApps:

- Масштабованість та Модульність: Ідеально підходить для складних DApps, які вимагають розширених функціональних можливостей та масштабованості.

- Строга Типізація з TypeScript: Поліпшує якість коду і спрощує управління великими проектами.

3. Сценарії Використання:

- Angular часто вибирають для великих корпоративних додатків і складних DApps, де важливі надійність та структурованість коду.

Next.js та Gatsby.js

Next.js являє собою потужний фреймворк, який змінює правила гри у світі веб-розробки, використовуючи переваги React - однієї з найпопулярніших JavaScript бібліотек. Завдяки своїй здатності до серверного рендерингу, Next.js не тільки забезпечує більш швидке завантаження сторінок, але й підвищує ефективність веб-додатків у відповідності до сучасних вимог пошукових систем [5]. Це робить його особливо привабливим для розробників, які прагнуть досягти високого рангу у пошукових системах, що є ключовим фактором для успіху багатьох онлайн бізнесів.

Однією з основних переваг Next.js є його гнучкість. Розробники можуть вибрати між повністю статичними сайтами, серверно-рендереними сторінками або комбінацією обох, в залежності від потреб проекту. Ця гнучкість робить Next.js ідеальним рішенням для широкого спектру веб-додатків, від простих блогів до складних комерційних платформ. Крім того, Next.js пропонує підтримку таких функцій, як автоматичне розбиття коду та оптимізоване завантаження, що дозволяє створювати високоефективні веб-додатки з кращим користувацьким досвідом.

Іншою важливою особливістю Next.js є легкість інтеграції з іншими інструментами та сервісами. Це включає широкі можливості для інтеграції з API, CMS, зовнішніми базами даних та іншими бекенд-сервісами, що робить його міцним фундаментом для комплексних веб-додатків. Додатково, підтримка різних

плагінів та модулів розширює функціональність Next.js, дозволяючи розробникам налаштувати його під специфічні потреби своїх проектів.

Next.js також відіграє значну роль у розвитку децентралізованих додатків (DApps), пропонуючи розробникам надійний інструмент для створення інтерфейсів користувача. З його допомогою можна ефективно інтегрувати веб-інтерфейси з блокчейн-технологіями, забезпечуючи високу продуктивність і зручність взаємодії для кінцевих користувачів.

Next.js, як фреймворк на базі React, займає особливе місце у світі сучасної веб-розробки, пропонуючи розробникам набір унікальних переваг, які роблять його одним з найбільш популярних виборів для створення веб-додатків. Серед основних аспектів, які відрізняють Next.js, варто відзначити його гнучкість, продуктивність та легкість у використанні, кожен з яких сприяє ефективній та інноваційній розробці веб-додатків.

Гнучкість

Next.js надає розробникам гнучкість у виборі між статичною генерацією сторінок (SSG) та серверним рендерингом (SSR). Це дозволяє використовувати оптимальний підхід для кожного конкретного випадку, роблячи фреймворк ідеальним для широкого спектру проектів, від легких лендінг-сторінок до складних ентерпрайз-рішень. Така гнучкість дозволяє враховувати індивідуальні потреби проекту, забезпечуючи найкращий користувацький досвід та оптимізацію продуктивності.

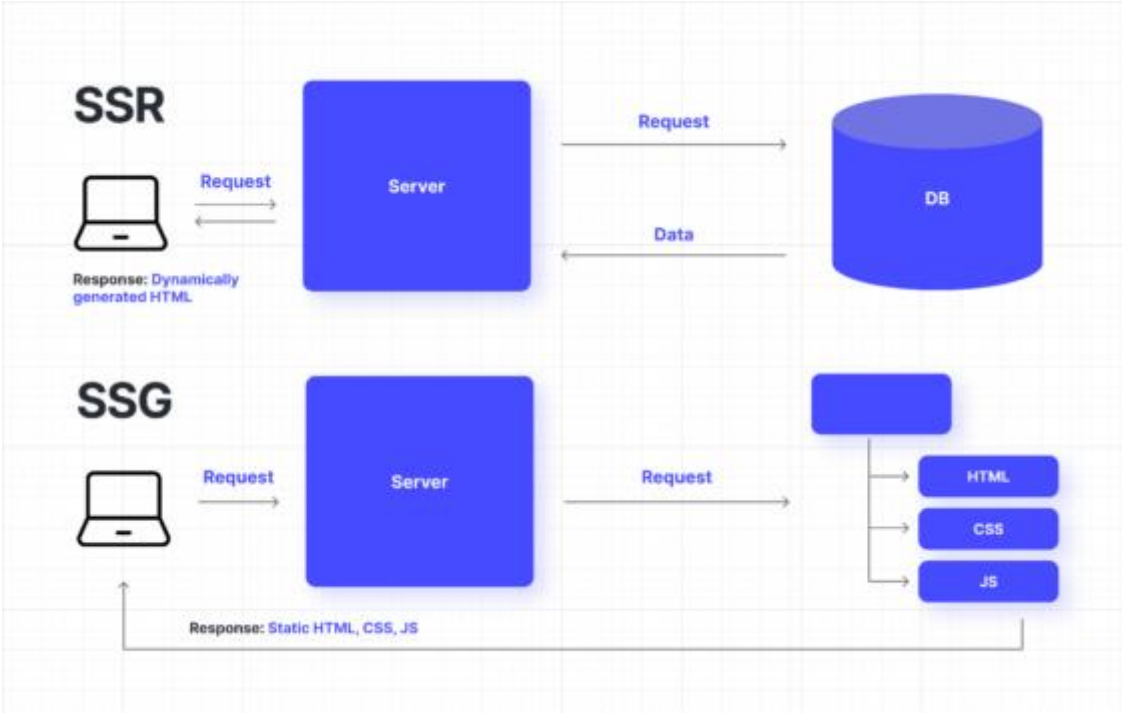



Рисунок 2.3 – Відмінності між SSR та SSG

Продуктивність

Next.js оптимізує продуктивність веб-додатків за рахунок автоматичного розбиття коду, ледачого завантаження компонентів і ефективного кешування. Це призводить до швидкого завантаження сторінок та підвищення загальної продуктивності додатка. Оптимізоване завантаження зображень, підтримка WebP формату та інтеграція з CDN додатково підвищують швидкість завантаження контенту, забезпечуючи відмінний користувацький досвід.

Легкість у Використанні

Next.js вирізняється легкістю у використанні, забезпечуючи розробникам простий та інтуїтивно зрозумілий інтерфейс. Вбудовані функції, такі як маршрутизація сторінок та підтримка API маршрутів, зменшують необхідність у використанні зовнішніх бібліотек, спрощуючи процес розробки. Розробники також оцінять підтримку hot-reloading, яка забезпечує миттєве оновлення контенту під час розробки, значно підвищуючи продуктивність та зручність роботи.

Сучасні Стандарти Розробки

Next.js дотримується найвищих стандартів сучасного веб-дизайну та

розробки. Він підтримує найсучасніші функції JavaScript та CSS, інтегрується з TypeScript та підтримує високий рівень безпеки. Ці функції роблять Next.js ідеальним вибором для створення високоякісних веб-додатків, які відповідають вимогам сучасного цифрового світу.

Next.js представляє собою фреймворк, який ідеально підходить для розробки сучасних веб-додатків, поєднуючи в собі гнучкість, продуктивність та легкість у використанні. Його здатність задовольнити широкий спектр вимог розробки, від статичних сайтів до складних веб-додатків, робить його одним з провідних інструментів у сфері веб-розробки. Завдяки цим якостям, Next.js є вибором багатьох професійних розробників, які прагнуть створити швидкі, ефективні та привабливі веб-додатки, які відповідають найсучаснішим стандартам.

Gatsby.js, відомий як статичний сайт-генератор для React, відкриває перед розробниками двері до ефективної та швидкої розробки веб-додатків. Використовуючи можливості сучасних веб-технологій, Gatsby.js дозволяє створювати веб-сайти, які не тільки швидко завантажуються, але й мають високий рівень взаємодії з користувачем. Цей фреймворк ідеально підходить для розробки статичних сайтів, блогів, портфоліо та інших веб-додатків, де важлива швидкість та візуальна привабливість [7].

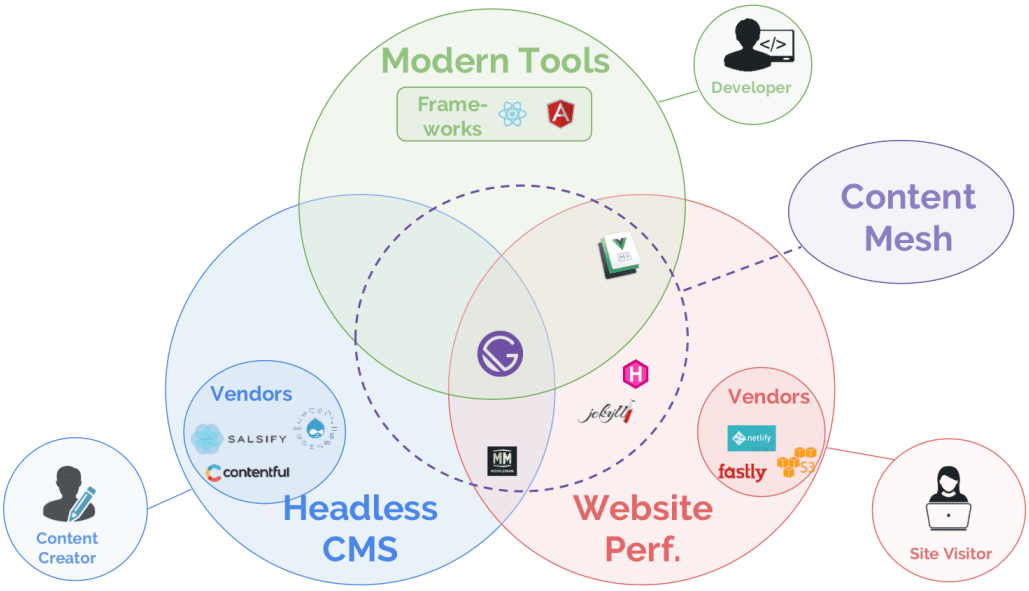



Рисунок 2.4 – Функціональна характеристика Gatsby.js

Однією з ключових особливостей Gatsby.js є його здатність інтегруватися з різними джерелами даних, такими як Markdown, CMS, RESTful API та GraphQL. Це надає розробникам гнучкість у виборі способів збору та використання контенту для своїх сайтів. Крім того, завдяки попередньому рендерингу сторінок на часі збірки проекту, Gatsby забезпечує високу швидкість завантаження, що є важливим фактором для SEO та загального користувацького досвіду. Gatsby також пропонує великий вибір плагінів, які роблять процес розробки ще більш гнучким та потужним. Від оптимізації зображень до інтеграції з аналітичними інструментами, ці плагіни дозволяють розширити функціональність сайтів без необхідності ручного кодування складних функцій.

Крім того, Gatsby пропонує відмінне середовище для розробки, інтегруючи такі технології, як hot-reloading, що дозволяє розробникам бачити зміни в реальному часі під час роботи над проектом. Це не тільки підвищує продуктивність розробки, але й забезпечує більш плавний та ефективний процес створення веб-сайтів.

У підсумку, Gatsby.js є ідеальним рішенням для розробників, які шукають швидкий, ефективний та гнучкий спосіб створення статичних веб-сайтів на базі React. Завдяки своїй швидкості, оптимізації для SEO та широким можливостям інтеграції, Gatsby відкриває широкі можливості для створення сучасних, високопродуктивних веб-сайтів, які відповідають вимогам сучасних користувачів інтернету.

Переваги для DApps:

У сучасному світі розробки децентралізованих додатків (DApps), важливість швидкості завантаження, ефективної взаємодії з користувачем і високої ефективності рендерингу сторінок не може бути недооцінена. Фреймворки, які використовуються у розробці DApps, грають ключову роль у забезпеченні цих аспектів. Особливо важливо, щоб ці фреймворки були здатні впоратися з динамічністю і мінливістю вимог, які характерні для DApps.

Перш за все, швидкість завантаження є критичним фактором, який впливає на користувацький досвід. У світі, де кожна секунда затримки може відштовхнути потенційних користувачів, здатність фреймворку швидко завантажувати і

відображати вміст є надзвичайно важливою. Це не тільки підвищує задоволення користувачів, але й сприяє кращому рейтингу веб-сайтів у пошукових системах.

Також важливою є ефективність взаємодії з користувачем. DApps часто включають складні взаємодії та транзакції, тому здатність фреймворку забезпечити плавні та інтуїтивно зрозумілі взаємодії є ключовою. Розробники повинні мати можливість створювати інтерфейси, які не просто привабливі візуально, але й легкі у використанні, надаючи користувачам чіткі вказівки та зворотний зв'язок під час виконання транзакцій або інших взаємодій.

Ефективність в рендерингу сторінок також є критичною. Фреймворки, що використовуються для розробки DApps, мають бути оптимізовані для швидкого рендерингу, забезпечуючи, щоб навіть найбільш вимогливі сторінки завантажувалися швидко та без помилок. Це стає особливо важливим у контексті блокчейн-технологій, де кожна транзакція або взаємодія може вимагати додаткових обчислень і взаємодії з блокчейном.

У підсумку, вибір правильного фреймворку для розробки DApps має вирішальне значення для успіху проекту. Фреймворк повинен не тільки відповідати технічним вимогам проекту, але й забезпечувати високий рівень користувацького досвіду, швидкість завантаження та ефективність взаємодії з користувачем. Такий підхід дозволяє створювати DApps, які не просто функціональні, але й зручні та привабливі для кінцевих користувачів.

При виборі інструментів для фронтенд-розробки децентралізованих додатків (DApps), розробники стикаються з різноманіттям варіантів, кожен з яких має свої унікальні характеристики та сфери застосування. Рішення про вибір між такими бібліотеками та фреймворками, як React.js, Vue.js, Angular, Next.js та Gatsby.js, залежить від специфіки проекту, його технічної складності, розміру команди, очікувань замовника та необхідних функціональних можливостей.

React.js відомий своєю гнучкістю та ефективністю, забезпечуючи стабільну основу для розробки інтерактивних інтерфейсів користувача. Його компонентний підхід дозволяє легко масштабувати та реорганізовувати проекти, що робить його ідеальним для різних видів DApps. Водночас, Vue.js пропонує простоту та легкість у використанні, що робить його привабливим для менших проектів або команд з меншим досвідом у фронтенд-розробці.

Angular, з іншого боку, є більш всебічним рішенням, що пропонує широкий спектр функціональностей, включаючи двосторонній байндінг, вбудовані сервіси та суворе керування станом. Цей фреймворк підходить для більш складних проектів, де необхідна висока надійність та структурований підхід до розробки.

Next.js та Gatsby.js, будучи фреймворками на основі React, приносять додаткові переваги у вигляді оптимізації для пошукових систем та кращої продуктивності завдяки серверному рендерингу та статичній генерації сторінок. Вони стають відмінним вибором для розробки DApps, де потрібні швидке завантаження та висока продуктивність.

Таким чином, процес вибору відповідної технології для фронтенд-розробки DApps вимагає глибокого аналізу потреб проекту та розуміння сильних та слабких сторін кожного із доступних інструментів. Вибір залежить від балансу між технічною складністю, необхідною швидкістю розвитку, очікуваннями користувачів та загальними цілями проекту.

# 3 ОПИС ПРОТОТИПУ

## 3.1 Підготовка до розробки проєкту

Для успішного втілення проєкту, планується створити прототип для онлайн-чату, який надаватиме користувачам можливість обмінюватися повідомленнями, а також здійснювати вхід в особисті акаунти через криптогаманець. Щодо вимог до інтерфейсу нашої інформаційної системи, вони формулюються таким чином:

1. Підтримка англійської мови є обов'язковою, оскільки це забезпечує широкий охоплення та доступність для глобальної аудиторії.

2. Адаптивний дизайн елементів інтерфейсу (таких як пункти меню, кнопки, поля введення, розкриваючі списки тощо) має бути налаштований так, щоб гарантувати оптимальне відображення та взаємодію на різних пристроях, враховуючи їхні розміри та найпоширеніші сценарії використання.

3. Дизайн інтерфейсу користувача розробляється з думкою про забезпечення працездатності системи навіть у випадку часткової втрати функціональності. Окремі елементи дизайну можуть мати спрощений вигляд, але доступ до основних функцій завжди має залишатися відкритим.

4. Інтерфейс системи повинен бути максимально простим та інтуїтивно зрозумілим для користувача, забезпечуючи можливість вирішення задач найшвидшим та найпростішим способом.

5. Система має бути розроблена так, щоб бути зрозумілою та легко освоюємою для користувачів без спеціальних технічних навичок, мінімізуючи потребу у зверненні до служби технічної підтримки.

Загалом, дизайн та функціональність інтерфейсу нашої системи мають відповідати високим стандартам сучасного користувацького досвіду, які формувалися під впливом найкращих прикладів існуючих сервісів у цій галузі.

Програмні вимоги.

Для розробки була обрана платформа Node.js версії 20.10.0, яка на момент роботи над магістерською роботою є LTS ( Long term support ) версією середовища

запуску програмного забезпечення, написаного за допомогою мови програмування JavaScript.

Середовище розробки

Для початку розробки попередньо було встановлено Visual Studio Code.

Visual Studio Code (VS Code) – це потужне, гнучке та високо налаштовуване середовище розробки (IDE), яке є ідеальним вибором для магістерських робіт у галузі програмування та розробки програмного забезпечення. Це безкоштовний та відкритий продукт від Microsoft, що підтримує множину мов програмування, включаючи, але не обмежуючись, JavaScript, Python, C++, Java та інші [12].

Основні переваги Visual Studio Code полягають у його легкості та високій продуктивності. Він не перевантажує систему, як це роблять деякі інші IDE, але при цьому надає широкий спектр функціональних можливостей. VS Code має вбудовану підтримку Git, що дозволяє легко керувати версіями коду прямо з середовища розробки. Це особливо корисно для магістерських проектів, де керування версіями та співпраця часто є ключовими.

Інтегрований термінал, великий вибір розширень і плагінів, що можна легко встановити, роблять VS Code дуже адаптивним і здатним задовольнити специфічні потреби магістерської роботи. Його інтуїтивний інтерфейс та гнучкість налаштувань дозволяють легко налаштовувати робоче середовище під конкретні завдання та переваги користувача.

Окрім цього, Visual Studio Code підтримує віддалену розробку, дозволяючи працювати з кодом, розміщеним на віддалених серверах, у контейнерах або в WSL. Це робить його ідеальним для проектів, що вимагають гнучкості у виборі розробницького середовища та віддаленої роботи.

Таким чином, Visual Studio Code є відмінним вибором для магістерських робіт, оскільки він надає все необхідне для ефективної розробки, від легкості та гнучкості до інтеграції з передовими інструментами та технологіями.

Необхідні елементи для побудови чату:

Як і будь-яка інформаційна система, веб-додаток у вигляді децентралізованого онлайн-чату складається з наступних, невід’ємних елементів, необхідних для її роботи:

- База даних
- Сервіс аутентифікації
- Сервер (для виконання CRUD-операцій: create, read, update, delete)
- Отримання повідомлень в режимі реального часу(websocket)
- Клієнтський інтерфейс – frontend.

У даній магістерській роботі для реалізації бази даних обрано використання технології блокчейну Subsocial, яка відрізняється своєю надійністю та безпекою. Блокчейн Subsocial є інноваційним рішенням, що забезпечує децентралізоване зберігання даних, ідеально підходить для цілей цього дослідження завдяки своїй високій пропускній здатності та гнучкості управління даними. Окрім того, для зберігання та завантаження зображень буде використано мережу IPFS, яка дозволяє зберігати великі обсяги даних, забезпечуючи високу швидкість доступу та надійність.

Сервіс авторизації в проекті буде побудовано на основі блокчейн-аккаунтів, що дозволяє використовувати всі переваги децентралізованої мережі, забезпечуючи високий рівень безпеки та приватності користувачів. Цей підхід до авторизації забезпечить надійний контроль доступу до даних та їхню цілісність.

Для оптимізації розробки та забезпечення ефективності роботи з даними, планується використовувати спеціалізоване API для блокчейн-модулів. Це API дозволить здійснювати CRUD (створення, читання, оновлення, видалення) операції, інкапсулюючи складну логіку взаємодії з блокчейн-мережею. Такий підхід спростить процес розробки, дозволяючи розробникам зосередитися на функціональності додатку, не витрачаючи час на реалізацію низькорівневих операцій з мережею.

Загалом, використання комбінації блокчейну Subsocial, мережі IPFS та спеціалізованого API для блокчейн-модулів створює сильну основу для розробки надійного, безпечного та високопродуктивного веб-додатку в рамках цієї магістерської роботи.

У рамках проекту планується створити клієнтський інтерфейс у вигляді веб-

додатку, розробленого на базі фреймворку Next.js. Вибір Next.js як основної платформи для розробки був зроблений після ретельного аналізу та порівнянь з іншими можливими рішеннями. Основними перевагами Next.js є його здатність ефективно інтегрувати гнучкість розробки, пропоновану React, зі стандартизацією ключових програмних компонентів. Це сприяє швидкій та якісній розробці програмних продуктів.

Next.js вирізняється своїми можливостями щодо server-side rendering (SSR) та static-site generation (SSG), що забезпечують високу швидкість завантаження та оптимізацію для пошукових систем. Це особливо важливо для забезпечення кращого користувацького досвіду та покращення видимості веб-додатку в інтернеті. Крім того, Next.js підтримує гаряче перезавантаження та автоматичну оптимізацію зображень, що є важливими факторами для зручності розробки та підвищення продуктивності веб-додатку.

Таким чином, обраний фреймворк Next.js дозволяє не тільки раціонально використовувати переваги React для створення динамічних та відгукових користувацьких інтерфейсів, але й забезпечує додаткові можливості для оптимізації та поліпшення загальної продуктивності веб-додатку.

Ініціалізація проекту у використанні фреймворку Next.js починається з команди `npx create-next-app`, яка виконується в командному рядку. Ця команда автоматизує багато початкових налаштувань: створює необхідні папки та файли, встановлює залежності і конфігурує базові команди для розробки. Такий підхід значно спрощує процес налаштування нового проекту, дозволяючи розробнику зосередитися на важливіших аспектах розробки.

У структурі проекту, папка `pages` містить файли, які відповідають за окремі сторінки веб-додатку. Файл `_app.js` виступає ядром проекту, визначаючи загальні налаштування та обгортки для всіх сторінок. Файл `index.js` є стартовою точкою або головною сторінкою веб-додатку.

Для розробки серверної частини в папці `pages > api` знаходяться прикладні інтерфейси (APIs), що дозволяють взаємодіяти з бекендом. Це забезпечує гнучкість у створенні бекенд-функціональності без необхідності використання додаткових серверів або налаштувань.

У папці `public` зберігаються статичні ресурси проекту, такі як зображення,

документи та інші файли, які мають бути доступні без обробки сервером. Папка `styles` містить глобальні стилі та CSS-файли, необхідні для оформлення більшості сторінок додатку.

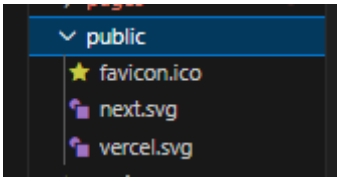



Рисунок 3.1 – вміст папки public

Файл `next.config.js` використовується для вказівки глобальних конфігурацій проекту, таких як налаштування оптимізації, підтримка локалізації та інші важливі параметри. А `package.json` зберігає ключову інформацію про проект, включаючи залежності, скрипти для запуску та тестування, а також інші метадані, які допомагають управляти проектом та його залежностями.

```
next.config.js > ...
1   /** @type {import('next').NextConfig} */
2   const nextConfig = {
3     reactStrictMode: true,
4     webpack(config) {
5       config.module.rules.push({
6         test: /\.svg$/i,
7         issuer: /\.[jt]sx?$/,
8         use: ['@svgr/webpack'],
9       })
10
11      return config
12    },
13  }
14
15  module.exports = nextConfig
16
```

Рисунок 3.2 – Код базової конфігурації проєкту на Next.js

Цей код є частиною конфігурації з’єднання з API Subsocial у проекті. На початку файлу імпортовано тип `SubsocialApi` з пакету `@subsocial/api`, що є важливим для типізації та використання функцій Subsocial API.

Оголошено інтерфейс `SubsocialConnectionConfig`, який визначає структуру конфігурації для з’єднання з Subsocial. Він містить поля `substrateUrl` та `ipfsNodeUrl`, які є обов’язковими, і визначають URL адреси для підключення до

Substrate та IPFS вузлів відповідно. Опціональні поля `ipfsAdminNodeUrl` та `postConnectConfig` дозволяють вказати альтернативну URL для IPFS адміністраторського вузла та функцію, яка виконується після підключення до API.

Далі оголошено константу `CRUST_TEST_AUTH_KEY`, яка є авторизаційним ключем і використовується тільки для тестових цілей у тестовій мережі.

`DEFAULT_PROD_CONFIG` - це об’єкт, який визначає стандартну конфігурацію для з’єднання з продакшн-версією Subsocial. Він містить URL адреси для Substrate та IPFS вузлів, а також опціональний URL для адміністраторського вузла IPFS. Функція `postConnectConfig` встановлює заголовки авторизації для запису в IPFS, використовуючи `CRUST_TEST_AUTH_KEY`.

На завершення, оголошено функцію `getConnectionConfig`, яка повертає об’єкт `DEFAULT_PROD_CONFIG`. Ця функція може використовуватися в інших частинах проекту для отримання конфігурації з’єднання з Subsocial.

Загалом, ця структура і процес ініціалізації проекту в Next.js забезпечує розробникам максимально зручний та ефективний шлях для створення сучасних веб-додатків, дозволяючи їм концентруватися на специфіці та функціональності продукту, а не на технічних деталях його реалізації.

У проекті буде створено глобальну папку `components`, яка буде служити сховищем для компонентів, що використовуються для побудови сторінок. В цій папці будуть організовані численні підкаталоги, кожен з яких міститиме основний файл конкретного компоненту та, за потреби, відповідні файли стилів.

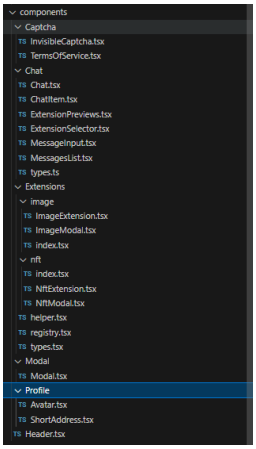



Рисунок 3.3 – Список створених компонентів

```tsx
import '@/styles/globals.css'
import type { AppProps } from 'next/app'
import { useEffect, useRef } from "react";
import { useMyAccount } from "@/stores/my-account";
import { SWRConfig } from 'swr';

export default function App({ Component, pageProps }: AppProps) {
  const isInitialized = useRef(false)

  useEffect(() => {
    if (isInitialized.current) return
    isInitialized.current = true
    useMyAccount.getState().init().catch(console.error)
  }, [])

  return <SWRConfig value={{ fallback: pageProps.fallback }}>
    <Component {...pageProps} />
  </SWRConfig>
}
```

Рисунок 3.4 – Код точки входу в проєкт (app.tsx)

```ts
const VERIFIER = 'https://www.google.com/recaptcha/api/siteverify'
const BURN_AMOUNT = 0.5 * 10 ** 10

async function getServerAccount() {
  const mnemonic = process.env.SERVER_MNEMONIC

  if (!mnemonic) {
    throw new Error('Invalid Mnemonic')
  }

  const keyring = new Keyring()
  await waitReady()
  return keyring.addFromMnemonic(mnemonic, {}, 'sr25519')
}

function getCaptchaSecret() {
    const secret = process.env.CAPTCHA_SECRET
    if (!secret) throw new Error('Invalid Captcha Secret')
return secret
}

async function getPaymentFee() {
  const signer = await getServerAccount()
  const subsocialApi = await getSubsocialApi()
  const substrateApi = await subsocialApi.substrateApi
  const paymentFee = await substrateApi.tx.energy
    .generateEnergy(signer.address, BURN_AMOUNT)
    .paymentInfo(signer.address)
  return paymentFee.partialFee.toNumber() + BURN_AMOUNT
}

async function isEnoughBalance() {
  const signer = await getServerAccount()
  const subsocialApi = await getSubsocialApi()
  const substrateApi = await subsocialApi.substrateApi
  const balance = await substrateApi.query.system.account(signer.address)
  const paymentFee = await getPaymentFee()
  return balance.data.free.toNumber() > paymentFee
}

async function verifyCaptcha(captchaToken: string) {
  const formData = new URLSearchParams()
  formData.append('secret', getCaptchaSecret())
  formData.append('response', captchaToken)
  const res = await fetch(VERIFIER, {
    method: 'POST',
    body: formData,
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
  })
  const jsonRes = await res.json()
  if (!jsonRes.success) throw new Error('Invalid Token')
  return true
}
```

Рисунок 3.5 – Код логіки взаємодії з Subsocial API

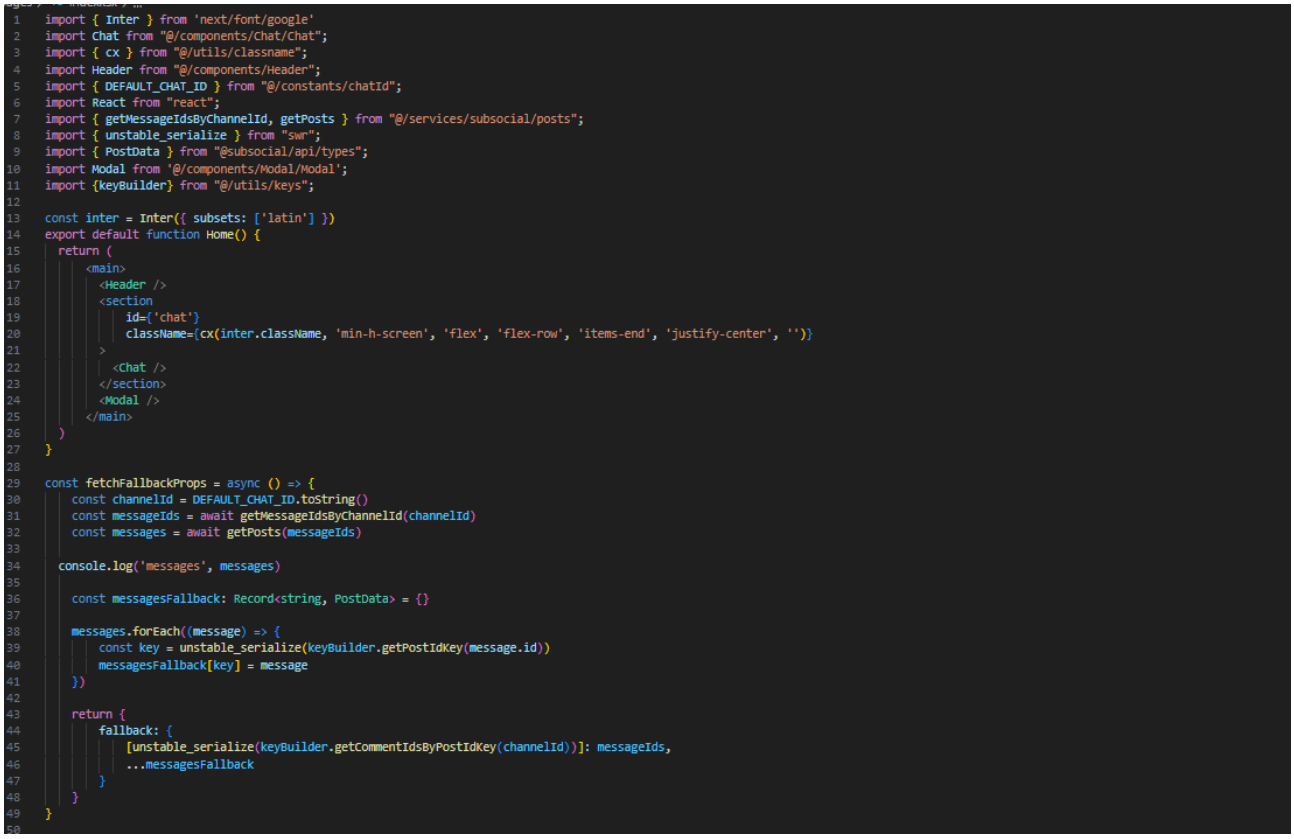
У цьому проекті буде використовуватися контейнер (layout) як ключовий компонент, що включатиме основні частини веб-сайту, такі як «шапка», навігація. Цей контейнер дозволить динамічно вставляти інші компоненти, забезпечуючи унікальність кожної сторінки. Першочергово розробляться основні елементи контейнера: «шапка» з навігаційним меню та.

components > TS Header.tsx > ...

```tsx
import { useMyAccount } from '@/stores/my-account'
import LogoSvg from '@/assets/icons/grill-logo.svg'

const Header = () => {
  const address = useMyAccount((state) => state.address)

  const login = useMyAccount((state) => state.login)
  const logout = useMyAccount((state) => state.logout)

  return (
    <div className="sticky top-0 z-20 flex h-14 items-center border-b border-border-gray bg-background-light bg-slate-800 border-gray-700">
      <div className="relative mx-auto w-full max-w-screen-md px-2 grid h-14 items-center py-1.5">
        <div className="flex items-center justify-between gap-4">
          <LogoSvg />
          {address ? (
            <span>
              <span className="mr-4">{address}</span>
              <button
                onClick={() => logout()}
                className="btn btn-outline border-indigo-600 hover:bg-transparent hover:ring-2 hover:text-white min-h-fit h-auto py-3 normal-case font-normal hover:border-indigo-700 px-5 flex-nowrap rounded-3xl"
              >
                Logout
              </button>
            </span>
          ) : (
            <button
              onClick={() => login()}
              className="btn min-h-fit h-auto py-3 px-5 hover:border-indigo-700 hover:bg-indigo-700 flex-nowrap normal-case font-normal rounded-3xl"
            >
              Login
            </button>
          )}
        </div>
      </div>
    </div>
  )
}

export default Header
```

Рисунок 3.6 – Код компоненту Header

Файл `pages/index.ts` у цьому проекті виконує роль головної сторінки веб-додатку. Він починається з імпортування необхідних залежностей та компонентів. Серед них, `Inter` з `next/font/google` для кастомного шрифту, компонент `Chat` з директорії `@/components/Chat/Chat` для реалізації чату, утиліта `cx` з `@/utils/classname` для управління класами CSS, `Header` з `@/components/Header` як компонент заголовка сторінки, константа `DEFAULT_CHAT_ID` з `@/constants/chatId` для визначення ідентифікатора чату, та кілька інших імпортів, включаючи `React`, функції з `@/services/subsocial/posts` для отримання повідомлень та `Modal` з `@/components/Modal/Modal`.

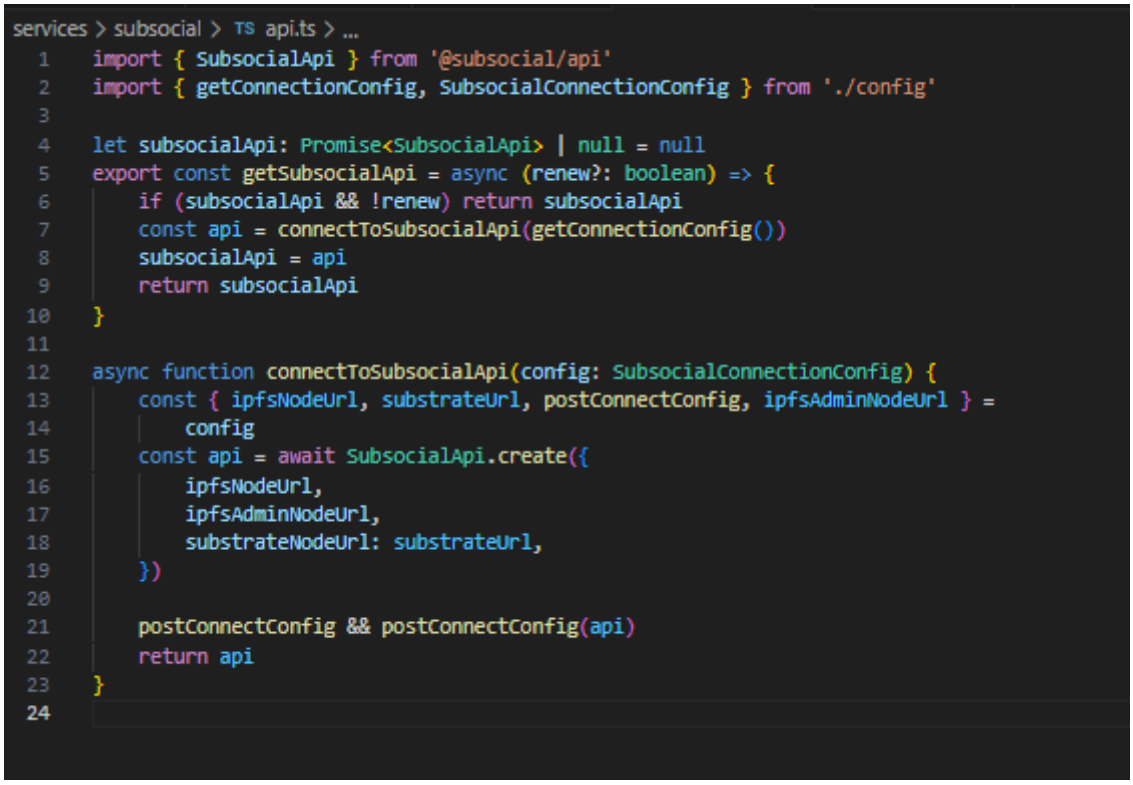


```tsx
import { Inter } from 'next/font/google'
import Chat from "@/components/Chat/Chat";
import { cx } from "@/utils/classname";
import Header from "@/components/Header";
import { DEFAULT_CHAT_ID } from "@/constants/chatId";
import React from "react";
import { getMessageIdsByChannelId, getPosts } from "@/services/subsocial/posts";
import { unstable_serialize } from "swr";
import { PostData } from "@subsocial/api/types";
import Modal from '@/components/Modal/Modal';
import {keyBuilder} from "@/utils/keys";

const inter = Inter({ subsets: ['latin'] })
export default function Home() {
  return (
      <main>
        <Header />
        <section
          id={'chat'}
          className={cx(inter.className, 'min-h-screen', 'flex', 'flex-row', 'items-end', 'justify-center', '')}
        >
          <Chat />
        </section>
        <Modal />
      </main>
  )
}

const fetchFallbackProps = async () => {
    const channelId = DEFAULT_CHAT_ID.toString()
    const messageIds = await getMessageIdsByChannelId(channelId)
    const messages = await getPosts(messageIds)

  console.log('messages', messages)

    const messagesFallback: Record<string, PostData> = {}

    messages.forEach((message) => {
        const key = unstable_serialize(keyBuilder.getPostIdKey(message.id))
        messagesFallback[key] = message
    })

    return {
        fallback: {
            [unstable_serialize(keyBuilder.getCommentIdsByPostIdKey(channelId))]: messageIds,
            ...messagesFallback
        }
    }
}
```

Рисунок 3.7 – Код модуля-контейнера index.tsx

У цьому файлі описано компонент `Home`, який повертає основний розмітковий блок `<main>`, що включає в себе `Header`, секцію для чату та модальне вікно `Modal`. Секція чату стилізована за допомогою утиліти `cx`, де використовується шрифт `Inter`, класи для мінімальної висоти екрану, відображення флекс-блоків та вирівнювання елементів.

Файл також містить асинхронну функцію `fetchFallbackProps`, яка використовується для попереднього завантаження даних чату. Вона отримує ідентифікатори повідомлень за допомогою `getMessageIdsByChannelId`, а потім завантажує ці повідомлення через `getPosts`. Кожне повідомлення обробляється та зберігається у об’єкті `messagesFallback` за ключем, який генерується функцією `unstable_serialize` з `keyBuilder.getPostIdKey`. Результатом роботи цієї функції є об’єкт з властивістю `fallback`, який містить ідентифікатори повідомлень та самі повідомлення, готові до використання у компоненті.



```
services > subsocial > TS api.ts > ...
1   import { SubsocialApi } from '@subsocial/api'
2   import { getConnectionConfig, SubsocialConnectionConfig } from './config'
3
4   let subsocialApi: Promise<SubsocialApi> | null = null
5   export const getSubsocialApi = async (renew?: boolean) => {
6       if (subsocialApi && !renew) return subsocialApi
7       const api = connectToSubsocialApi(getConnectionConfig())
8       subsocialApi = api
9       return subsocialApi
10  }
11
12  async function connectToSubsocialApi(config: SubsocialConnectionConfig) {
13      const { ipfsNodeUrl, substrateUrl, postConnectConfig, ipfsAdminNodeUrl } =
14          config
15      const api = await SubsocialApi.create({
16          ipfsNodeUrl,
17          ipfsAdminNodeUrl,
18          substrateNodeUrl: substrateUrl,
19      })
20
21      postConnectConfig && postConnectConfig(api)
22      return api
23  }
24
```

Рисунок 3.8 – Код підключення до мережі Subsocial

Цей код визначає процес створення та управління з’єднанням з API Subsocial у проекті. Спочатку він імпортує необхідні компоненти: `SubsocialApi` з `@subsocial/api` та функції `getConnectionConfig` та `SubsocialConnectionConfig` з локального файла конфігурації `./config`.

Оголошено змінну `subsocialApi`, яка ініціалізована як `null` і призначена для зберігання promise, що повертається від API Subsocial. Експортована функція `getSubsocialApi` є асинхронною та приймає параметр `renew`, який визначає, чи потрібно відновити з’єднання з API. Якщо змінна `subsocialApi` вже існує і `renew` не вказано, функція просто повертає поточний об’єкт API. В іншому випадку, вона викликає функцію `connectToSubsocialApi`, передаючи їй конфігурацію з `getConnectionConfig`, і зберігає результат у змінній `subsocialApi`.

Функція `connectToSubsocialApi` відповідає за безпосереднє підключення до Subsocial API. Вона приймає об’єкт конфігурації `SubsocialConnectionConfig`, з якого дістає необхідні параметри: URL IPFS вузла, URL Substrate вузла, опціональну функцію `postConnectConfig`, та URL адміністраторського IPFS вузла. Далі викликається метод `create` з `SubsocialApi`, передаючи необхідні параметри для ініціалізації з’єднання. Після створення API, якщо функція `postConnectConfig`

визначена, вона виконується з передачею щойно створеного API як аргументу.

В цілому, цей код демонструє впорядкований та ефективний спосіб ініціалізації та керування з'єднанням з Subsocial API, дозволяючи легко оновлювати або повторно використовувати це з'єднання в усьому проекті.

services > subsocial > TS energy.ts > ...

```ts
import { useMyAccount } from '@/stores/my-account'
import { useCallback, useEffect, useRef, useState } from 'react'

export default function useWaitHasEnergy() {
    const address = useMyAccount((state) => state.address)
    const energy = useMyAccount((state) => state.energy)
    const previousEnergy = useRef(energy)

    useEffect(() => {
        previousEnergy.current = energy
    }, [energy])

    const hasEnergyResolver = useRef<() => void | undefined>()

    const generatePromise = useCallback(
        () =>
            new Promise<void>((resolve) => {
                hasEnergyResolver.current = () => resolve()
            }),
        []
    )
    const [hasEnergyPromise, setHasEnergyPromise] =
        useState<Promise<void>>(generatePromise)

    useEffect(() => {
        setHasEnergyPromise(generatePromise())
    }, [address, generatePromise])

    useEffect(() => {
        if (energy && energy > 0) {
            if (hasEnergyResolver.current) {
                const resolveHasEnergy = () => {
                    hasEnergyResolver.current?.()
                    hasEnergyResolver.current = undefined
                }
                resolveHasEnergy()
            }
        } else {
            setHasEnergyPromise(generatePromise())
        }
    }, [energy, generatePromise, previousEnergy])

    return () => hasEnergyPromise
}
```

Рисунок 3.9 – Інкапсуляція логіки в React-хуку

Файлова структура

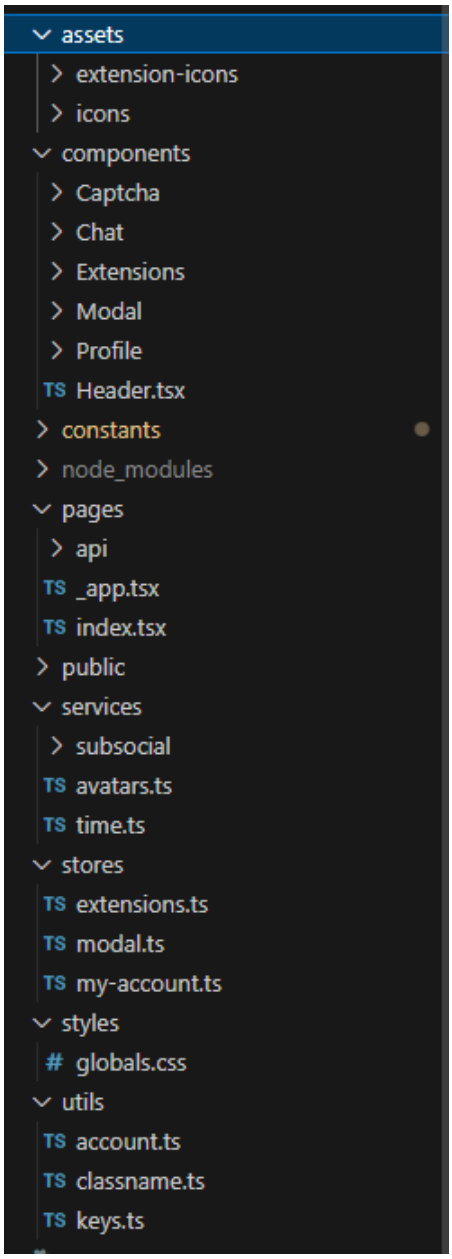



Рисунок 3.10 – Файлова структура

На зображенні зображено структуру проекту Next.js з використанням типової для цього фреймворку організації файлів та папок. Детальний опис такий: каталог `assets` може використовуватися для зберігання статичних ресурсів, які включають графічні зображення, які необхідні для додатку. У цьому каталозі є дві підпапки, `extension-icons` та `icons`, що натякає на те, що тут зберігаються іконки, які можуть використовуватися для різних розширень та інтерфейсних елементів відповідно. Каталог `components` містить кілька підпапок, що відповідають за різні реюзабельні компоненти в рамках додатку. Наприклад, підпапка `Captcha` може

містити компоненти для відображення та обробки капчі; `Chat` для функціоналу чату; `Extensions` може включати компоненти, які розширюють базовий функціонал; `Modal` для модальних вікон; `Profile` для компонентів, що відображають або обробляють профілі користувачів. Файл `Header.tsx` відповідає за заголовок веб-сторінки, що може містити навігаційні лінки, лого та інші елементи заголовка.

Папка `constants` містить файли з константами, які визначають сталі дані чи конфігурації, що використовуються у декількох частинах програми.`node_modules` — це стандартна папка для Node.js проектів, що містить всі бібліотеки та залежності, які встановлюються через систему управління пакетами, таку як npm або Yarn.`pages` є ключовим каталогом для Next.js, оскільки кожен файл у цій папці автоматично стає маршрутом в додатку. Файл `_app.tsx` є спеціальним компонентом, що обгортає всі інші сторінки та може використовуватися для визначення глобальних стилів та загальної логіки. `index.tsx` представляє головну сторінку веб-сайту.

У каталозі `services` є файли `avatars.ts` та `time.ts`, які, ймовірно, містять логіку обслуговування, пов'язану з аватарами користувачів та часовими функціями відповідно.

`stores` може використовуватися для управління станом додатку, з файлами, такими як `extensions.ts` для зберігання стану розширень, `modal.ts` для модальних вікон та `my-account.ts` для даних облікового запису користувача.

`globals.css` у папці `styles` ймовірно містить глобальні CSS-стилі, які застосовуються до всього додатку для забезпечення консистентності візуального вигляду.

Каталог `utils` містить утиліти або допоміжні інструменти, такі як `account.ts` для функцій, пов'язаних з обліковими записами, `classname.ts` для генерації імен класів та `keys.ts` для управління ключами або конфігураційними параметрами.

## 3.2 Опис готового проєкту

Фінальна версія проекту може значно відрізнятися від поточного стану додатку, проте це не знижує значення того, що додаток вже містить основний функціонал, який відповідає встановленим вимогам до інформаційних систем. У цьому розділі представлено огляд готових функцій інформаційної системи та демонстрацію її візуального аспекту. На ілюстрації, розміщеній нижче, показано базову сторінку додатку у момент першого використання користувачем.

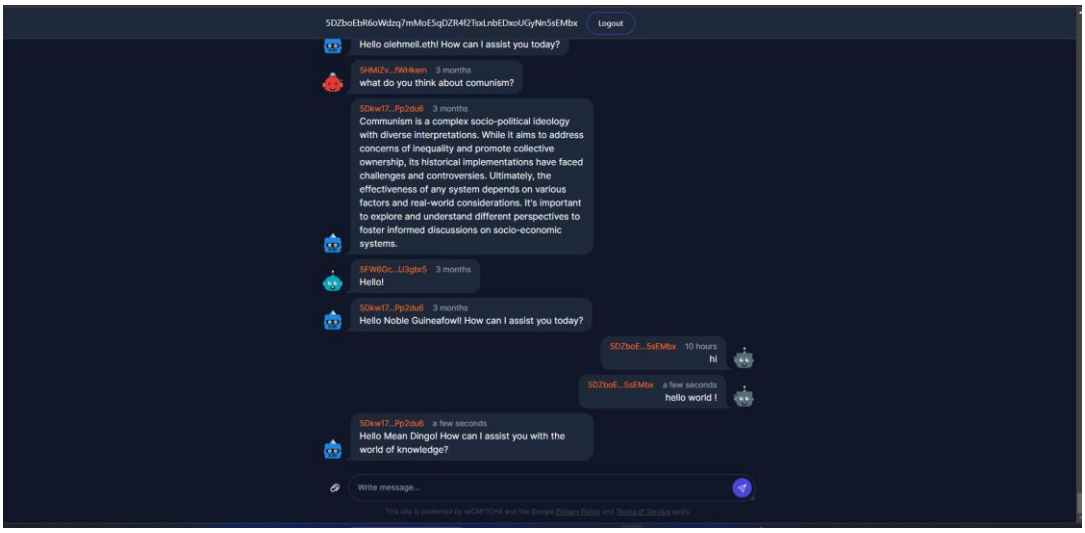



Рисунок 3.11 – Головна сторінка додатку

У поточному варіанті проекту для користувача доступна тільки одна сторінка, яка використовується для спілкування в чаті. Це основний функціонал, який було реалізовано на даному етапі. Мова інтерфейсу додатку на момент розробки обмежена англійською, що робить його доступним для широкого кола міжнародних користувачів, але також може обмежувати його зручність для носіїв інших мов.

Додатково, в додатку передбачений функціонал завантаження ілюстрацій. Користувачі мають можливість завантажувати зображення прямо у чат, що дозволяє їм ділитися візуальним контентом. Цей функціонал може бути особливо корисним у контексті обговорення творчих проектів, подій або навіть просто для особистісного вираження.

Крім того, в інтерфейсі передбачена можливість завантаження NFT (невзаємозамінних токенів), що є сучасною та інноваційною функцією. Це дозволяє

користувачам взаємодіяти з цифровими активами, які представляють цінність в цифровому світі, і може відкрити нові можливості для використання додатку в майбутньому.

На ілюстрації, розміщеній нижче, можна побачити візуальне представлення цього додаткового функціоналу, що демонструє, як зображення та NFT можуть бути інтегровані в чат. Це забезпечує більш глибокий та багатогранний досвід спілкування в додатку.

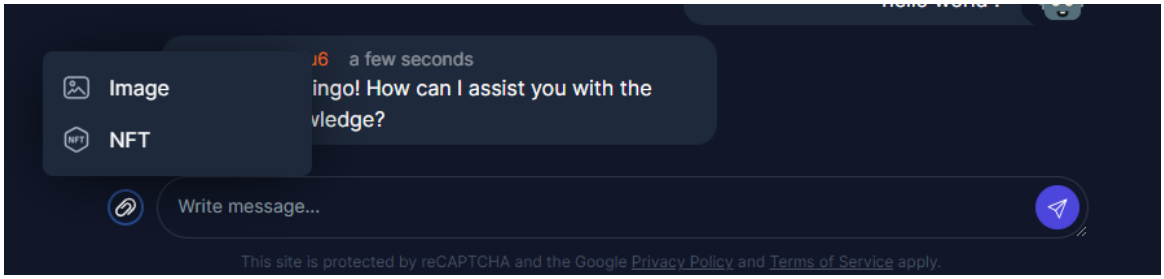



Рисунок 3.12 – Модальне вікно
додавання нової транзакції

На ілюстрації, розміщеній нижче, представлено інтерфейс чату для ситуації, коли користувач не увійшов у чат.

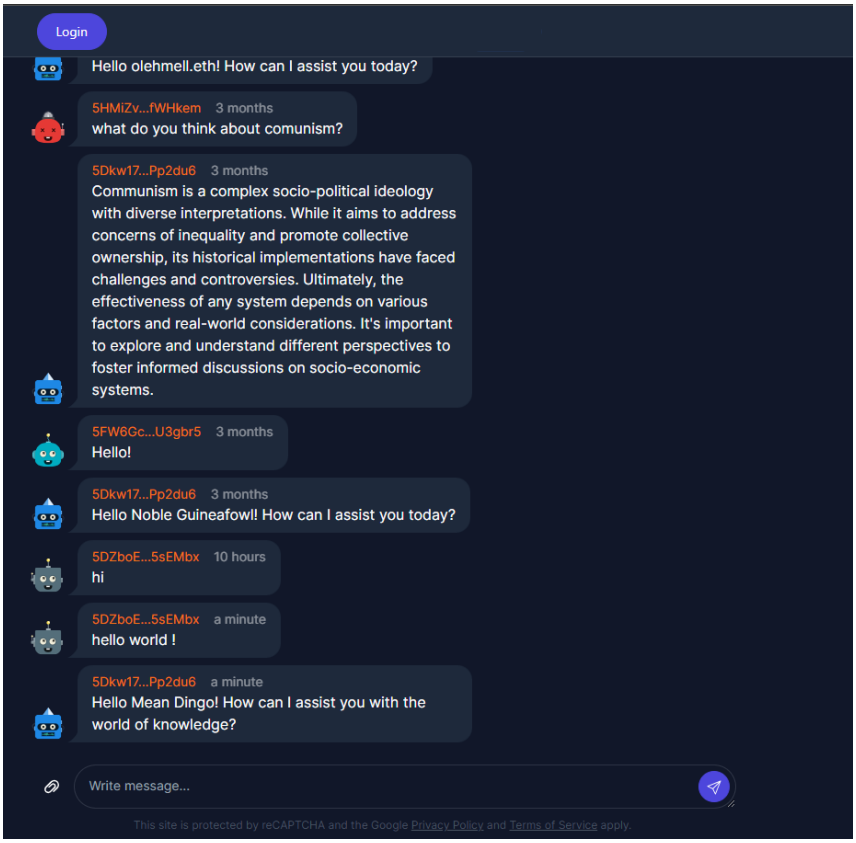



Рисунок 3.13 – Модальне вікно
додавання нової транзакції

# ВИСНОВОК

У ході виконання проєкту були досліджені: децентралізовані технології в контексті вебу та випробувані методи створення Web-орієнтованої інформаційної системи. Проведено дослідження ефективності використання різних технологій для реалізації функціонального і готового до експлуатації Web-додатку. Для цього було проаналізовано переваги програмного забезпечення, фреймворків та засобів, які допомагають полегшити розробку.

Також було розглянуто певні перспективи використання технологій, , описано певний перелік можливих варіантів використання, їх недоліки та переваги.

Розглянуті та використані технології показали, що раціональний вибір технологій для розробки проєкту може полегшити процес розробки.

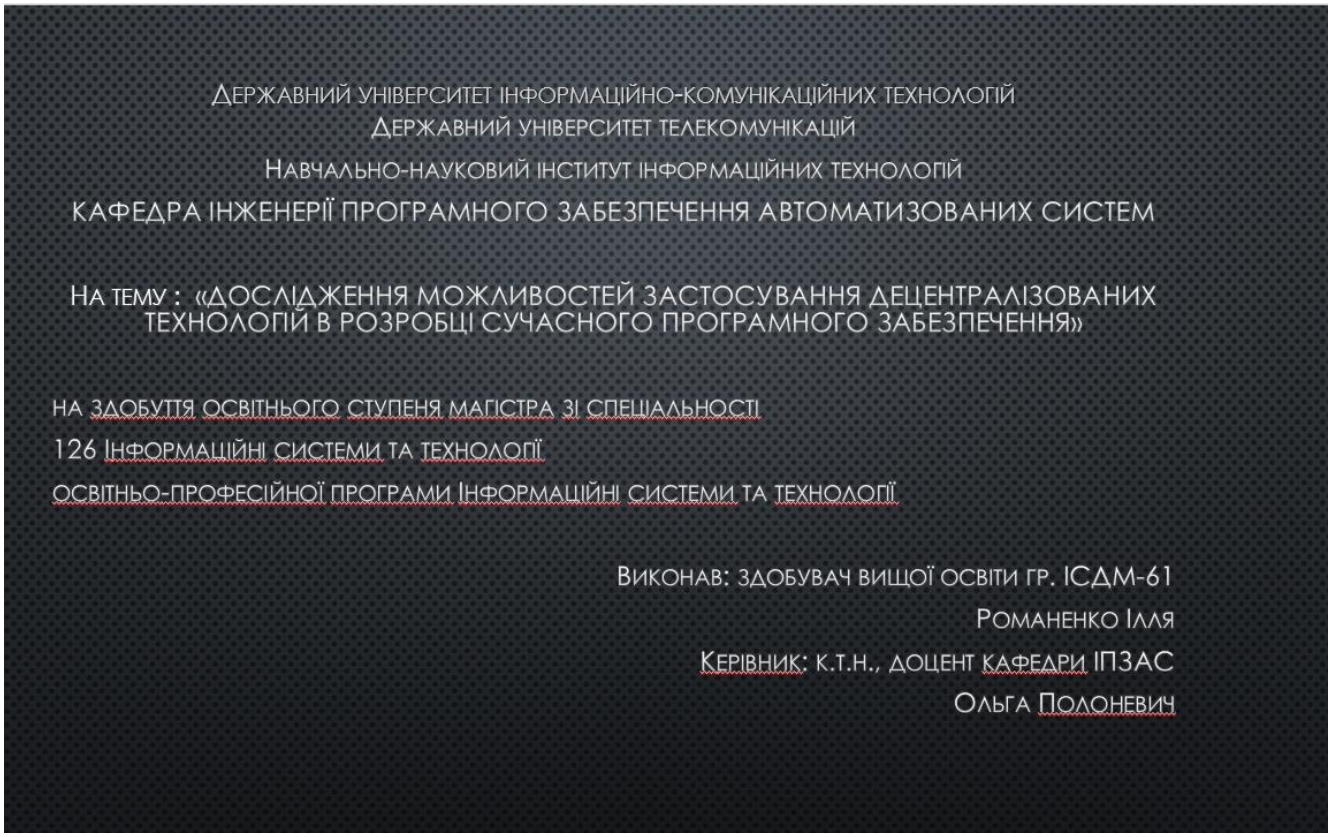
У результаті даної магістерської роботи було створено прототип web-додатку, який можна використовувати в якості навчальної бази для отримання досвіду роботи з інтеграцією децентралізованих технологій для веб-орієнтованого програмного забезпечення або за призначенням як децентралізований чат.

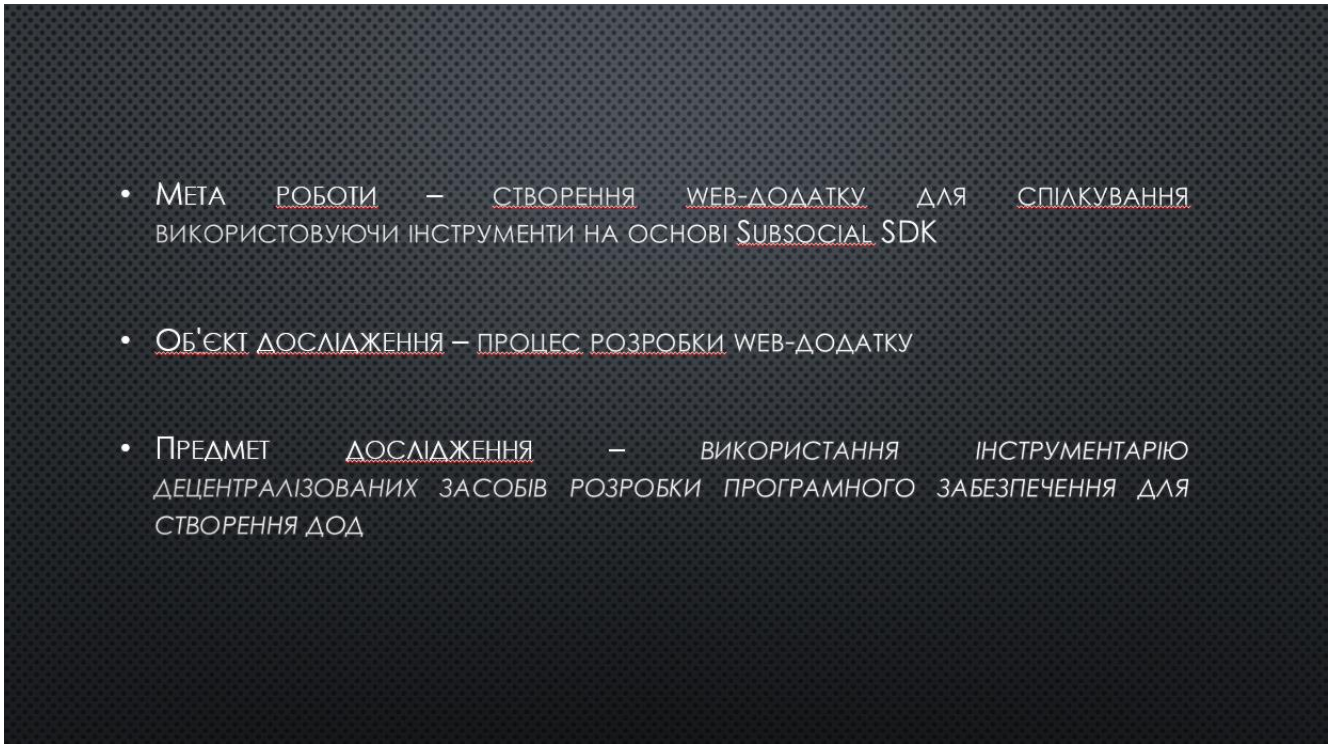
В ході виконання цієї роботи було досліджено сучасні інструменти та методики розробки, визначено їхні переваги та недоліки, а також проведено порівняльний аналіз різних технологій. В якості основи вибрано мову програмування JavaScript, яка є домінуючою у сфері веб-розробки. Для створення інформаційної системи використовувався фреймворк Next.js разом з рядом додаткових бібліотек, сумісних з ним. Завдяки використанню Next.js, система отримала відмінну SEO-оптимізацію, що сприяє підвищенню її індексації та ранжуванню в пошукових системах, що, у свою чергу, відіграє ключову роль у приваблені нових клієнтів.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Павло Кравченко, Богдан Скрябін, Оксана Дубініна "Блокчейн і децентралізовані системи. Частина 1"
2. React [Електронний ресурс]. – Режим доступу: https://react.dev/
3. Документація Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org/
4. Документація Angular [Електронний ресурс]. – Режим доступу: https://angular.io/
5. IPFS is the Distributed Web. js [Електронний ресурс]. – Режим доступу: https://ipfs.io
6. Документація Next.js [Електронний ресурс]. – Режим доступу: https://nextjs.org
7. Документація Gatsby.js [Електронний ресурс]. – Режим доступу: https://www.gatsbyjs.com/
8. Документація Cosmos SDK [Електронний ресурс]. – Режим доступу: https://v1.cosmos.network/sdk
9. Документація Web3.js [Електронний ресурс]. – Режим доступу: https://v1.cosmos.network/sdk
10. Документація Subsocial SDK [Електронний ресурс]. – Режим доступу: https://docs.subsocial.network/docs/develop/sdk/installation/
11. Документація Polkadot.js [Електронний ресурс]. – Режим доступу: https://polkadot.js.org/docs/api/
12. Документація Visual Studio Code [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Visual_Studio_Code

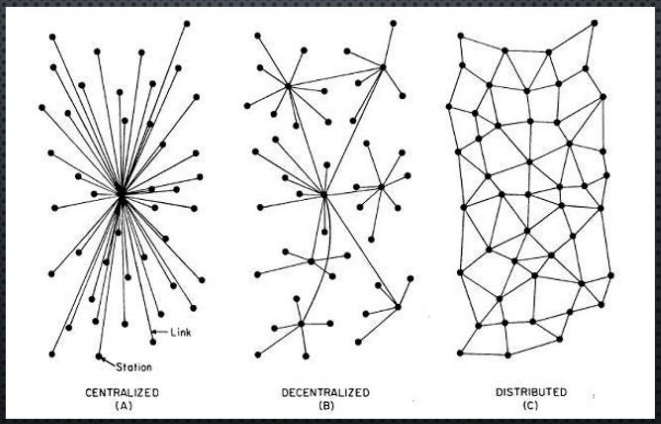
# ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)



- Мета роботи – створення web-додатку для спілкування використовуючи інструменти на основі Subsocial SDK
- Об'єкт дослідження – процес розробки web-додатку
- Предмет дослідження – *використання інструментарію децентралізованих засобів розробки програмного забезпечення для створення дод*

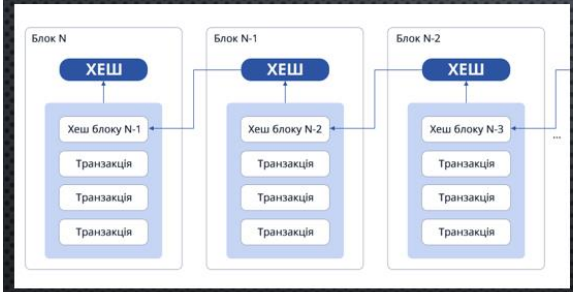
# ОСОБЛИВОСТІ МЕРЕЖ



# ДЕЦЕНТРАЛІЗОВАНІ ТЕХНОЛОГІЇ

- Децентралізовані технології — це широкий клас систем, які дозволяють розподіл обчислювальних та управлінських ресурсів серед різноманітних вузлів або агентів, а не зосереджують їх у одному центральному місці. Ця концепція передбачає, що кожен вузол у такій системі здатний незалежно обробляти дані та приймати рішення, що сприяє збільшенню надійності, безпеки та ефективності загалом.

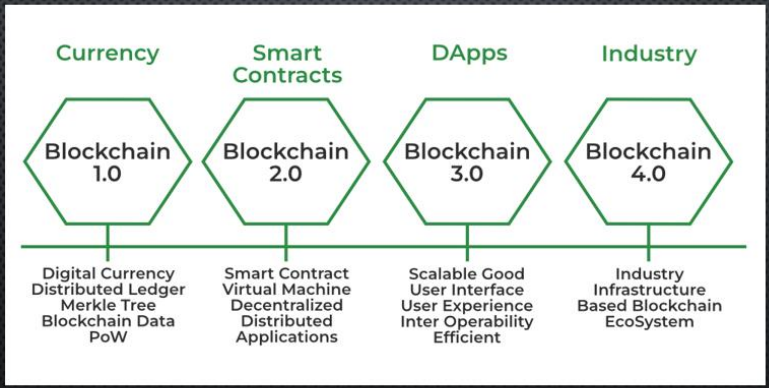
# БЛОКЧЕЙН




- Блокчейн як структура даних: послідовно пов'язаний ланцюг блоків, що містять криптографічно захищені записи. Кожен блок містить пакет транзакцій та інформацію про попередній блок, що забезпечує незмінність і безперервність даних.

- Блокчейн у загальному значенні: Блокчейн - це децентралізована технологія, яка дозволяє безпечне ведення розподіленого реєстру транзакцій або інших даних, гарантуючи прозорість, безпеку та неможливість підробки записів.

# Еволюція та покоління блокчейнів




# DAPPS

- Що таке DApps (децентралізовані додатки) - це програми, що функціонують на блокчейні або розподілених мережах.

- Безпека і прозорість: DApps забезпечують високий рівень безпеки та прозорості завдяки блокчейн технології.

- Автономність: Вони не контролюються жодною централізованою організацією, забезпечуючи повну автономію користувачам.

- Токенізація : Більшість DApps використовують токени для інцентивів та взаємодії у своїх екосистемах.

# NEXT.JS

У рамках проекту планується створити клієнтський інтерфейс у вигляді веб-додатку, розробленого на базі фреймворку Next.js. Вибір Next.js як основної платформи для розробки був зроблений після ретельного аналізу та порівнянь з іншими можливими рішеннями. Основними перевагами Next.js є його здатність ефективно інтегрувати гнучкість розробки, пропоновану React, зі стандартизацією ключових програмних компонентів. Це сприяє швидкій та якісній розробці програмних продуктів

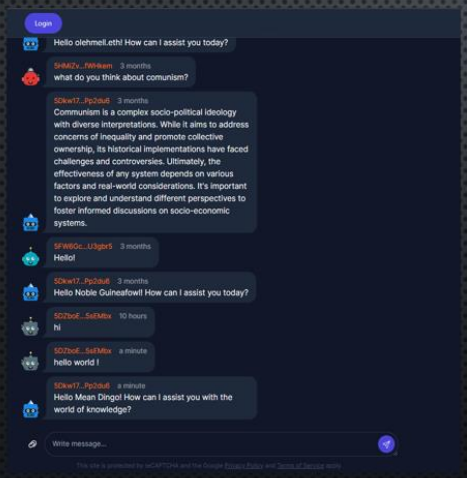
# ПЕРЕВАГИ ОБРАНОГО ІНСТРУМЕНТАРІЮ

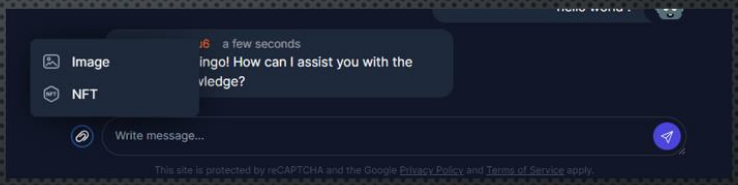
- Маршрутизація на основі файлової системи: Легке створення маршрутів через директорію `pages`.
- Підтримка API маршрутів Ідеально для повноцінних веб-додатків.
- Вбудовані CSS та Sass Легке імпортування стилів у компоненти.
- Підтримка TypeScript Для більш надійного коду.
- Активна спільнота
- Багато ресурсів та інтеграцій.

# ЕЛЕМЕНТИ ПРОЄКТУ

- БАЗА ДАНИХ (БЛОКЧЕЙН SUBSOCIAL І IPFS)
- СЕРВІС АУТЕНТИФІКАЦІЇ (БЛОКЧЕЙН-АККАУНТИ)
- СЕРВЕР (ДЛЯ ВИКОНАННЯ CRUD-ОПЕРАЦІЙ: CREATE, READ, UPDATE, DELETE) (SUBSOCIAL API)
- ОТРИМАННЯ ПОВІДОМЛЕНЬ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ(WEBSOCKET)
- КЛІЄНТСЬКИЙ ІНТЕРФЕЙС – FRONTEND.(NEXT.JS)

# ПРИКЛАД ІНТЕРФЕЙСУ



# ВИСНОВОК

- У ХОДІ ВИКОНАННЯ МАГІСТЕРСЬКОГО ПРОЕКТУ БУЛИ ДОСЛІДЖЕНІ: ЗАСТОСУВАННЯ ДЕЦЕНТРАЛІЗОВАНИХ ТЕХНОЛОГІЙ В СФЕРІ ВЕБ ТА ЇХ ПЕРСПЕКТИВИ
- ПРОВЕДЕНО ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ РІЗНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ФУНКЦІОНАЛЬНОГО І ГОТОВОГО ДО ЕКСПЛУАТАЦІЇ WEB-ДОДАТКУ. ДЛЯ ЦЬОГО БУЛО ПРОАНАЛІЗОВАНО ПЕРЕВАГИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ФРЕЙМВОРКІВ ТА ЗАСОБІВ, ЯКІ ДОПОМАГАЮТЬ ПОЛЕГШИТИ РОЗРОБКУ.