Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА СОЦІАЛЬНОЇ МЕРЕЖІ З МОЖЛИВІСТЮ НАДСИЛАННЯ КРИПТОВАЛЮТИ МІЖ КОРИСТУВАЧАМИ З ВИКОРИСТАННЯМ WEB3- ТА WEB5-ТЕХНОЛОГІЙ ДЛЯ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ ІНФОРМАЦІЇ КОРИСТУВАЧІВ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1
Сердінов Б.А.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сердінову Богдану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка соціальної мережі з можливістю надсилання криптовалюти між користувачами з використанням Web3- та Web5-технологій для забезпечення інтелектуальної власності інформації користувачів

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 02 червня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі,документація мови Goland, DWN, DID, Postgres.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд технологій Web3 та Web5.
2. Огляд існуючих децентралізованих соціальних мереж.
3. Аналіз основного функціоналу децентралізованої соцмережі.
4. Вибір архітектури для побудови високонавантаженого проєкту.
5. Розробка децентралізованої соціальної мережі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність створення децентралізованої соціальної мережі, постановка задачі, тестові зображення.

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Вибір архітектури проєкту	21.04.23-30.04.23	
5	Дослідження Web3 та Web5 технологій	01.05.23-06.05.23	
6	Програмна реалізація	06.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-26.05.23	
8	Перевірка на плагіат	27.05.23	
9	Рецензування	28.05.23	
10	Підготовка презентації та доповіді	29.05.23-30.05.23	
11	Занесення роботи в електронний архів	06.06.23	
12	Попередній захист кваліфікаційної роботи	11.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ _____доц. Вечірська І.Д._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи містить: 60 с., 15 рисунків, 30 джерел.

ВЕБЗАСТОСУНОК, СУБД, РЕЛЯЦІЙНА БАЗА ДАНИХ, СУТНІСТЬ, НОРМАЛІЗАЦІЯ ДАНИХ, МОВА ЗАПИТІВ SQL, POSTGRES, DOCKER, GOLAND IDE, GOLANG, DWN, DID, LIGHTNING.

Об’єктом роботи є децентралізований процес передачі даних між користувачами соціальної мережі на основі технологій Web3 та Web5 технологій з використанням DID та DWN.

Метою роботи є побудова соціальної мережі за допомогою сучасних технологій DID та DWN, які допоможуть створити таку систему, що надає людям можливість безпечно зберігати приватну інформацію, яка буде закодована за допомогою децентралізованого ідентифікатора, інформація не буде проаналізована сервісом за рахунок децентралізованого зберігання, а надає користувачам можливість самим керувати даними.

У результаті роботи зроблена програмна реалізація MVP децентралізованої соціальної мережі.

WEB-APPLICATION, DBMS, ENTITY, DATA NORMALIZATION, SQL, POSTGRES, DOCKER, GOLAND IDE, GOLANG, DWN, DID, LIGHTNING.

The object of the work is the decentralized process of data transfer between social network users based on Web3 and Web5 technologies using DID and DWN.

The purpose of the work is to create a social network using modern technologies that will help create such a system that gives people the opportunity to safely save their private information, which will not be aggregated by the service in order to influence and sell the quality of an opinion or things in any way, and one of the most important goals – creating such a social network that will not hold people hostage to the system, but will provide opportunities to transfer their data and resources to their domain.

As a result of the work, a software implementation of MVP of a decentralized social network was made.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DID – Decentralized Indentificator (децентралізований ідентифікатор)

DWN – Decentralized Web Nodes (децентралізований вебвузол)

MVP – Minimum Viable Product (мінімально життєздатний продукт)

Front-end – клієнтська частина застосунку

Back-end – серверна частина проєкту

Memonic phrase – фраза, яка складається з слів, довжиною 12, 18 чи 24

ВСТУП

Люди користуються соціальними мережами з різних причин, оскільки ці платформи надають широкий спектр можливостей і переваг. Соціальні мережі дозволяють людям зберігати зв’язок зі своїми родичами, друзями, колегами та іншими людьми навколо світу.

Вони надають зручні інструменти для обміну повідомленнями, коментування записів, спільного перегляду фотографій і відео, а також для організації подій і зустрічей. Соціальні мережі дозволяють людям висловлювати свої думки, ідеї, інтереси і становище на різні теми. Вони можуть ділитися своїми досягненнями, творчими роботами, фотографіями, відео і іншими контентом, щоб показати себе і свою особистість. Багато людей використовують соціальні мережі для отримання актуальної інформації, новин, трендів і подій. Вони підписуються на сторінки та групи, щоб отримувати оновлення від улюблених медіа, брендів, впливових осіб та інших джерел.

Соціальні мережі також використовуються для розваги і відпочинку. Люди переглядають фотографії, відео, меми, гумористичні контент і інші розважальні матеріали, які розповідаються у соціальних мережах. Вони можуть грати в ігри, долучатися до груп і спільнот зі спільними інтерес.

На сьогоднішній день існує велика кількість соціальних мереж, і їх число продовжує зростати з кожним роком. Однак точне число соціальних мереж у світі важко визначити через постійне появлення нових платформ та зникнення деяких існуючих. Але ці соціальні мережі є централізовані та не пропонують жодних напрямів у розвитку індустрії, тому з’явилась ідея інтегрувати таку монументальні річ як соціальні мережі, з новими напрямками у індустрії як Web3 та Web5 технології.

Web3 це технологічний стек, що ґрунтується на блокчейні, розподілених системах та криптографії. Web3 дозволяє створювати

децентралізовані проєкти, які забезпечують більшу приватність, безпеку та контроль користувачів над своїми даними.

Інтеграція Web3 у соціальні мережі відкриває нові можливості для користувачів та додає багато цінності до платформи.

За допомогою Web3 технологій, соціальні мережі можуть наблизитися до принципів децентралізації. Це означає, що користувачі мають більший контроль над своїми даними та приватністю. Вони можуть власноруч управляти своїми цифровими активами, дозволяючи безпосередньо спілкуватися з іншими користувачами та керувати своїм вмістом.

Web3 технології дозволяють використовувати криптовалюти для мікроплатежів та винагороди користувачів за їх вміст. Користувачі можуть отримувати винагороду за свої пости, коментарі, підписки або за участь у спільноті, а також надсилати платежі один одному.

Інтеграція Web3 у соціальні мережі може дозволити користувачам створювати та обмінювати власні цифрові активи, такі як NFT (невзаємозамінні токени) [1–5].

У останні роки було придумано також Web5 технологію, яка допомагає вирішити проблемі пов’язану із власністю даних та своєю цифровою ідентичністю, яка може використовуватись на будь якому інтернет ресурсі, та може виходити за рамки соціальних мереж, за допомогою Web5 юзер може переносити свої дані у блокчейн, де вони будуть захищені та розподілені по мережі.

Актуальність роботи полягає у створенні соціальної мережі, яка буде використовувати Web3 технології для переказів криптовалюти між юзерами, та технологія Web5 для того, щоб забезпечувати інтелектуальну власність даними, якими володіє юзер. А також було прийнято рішення використовувати найкраще, що надає централізовані соціальні мережі, такі як Twitter, Instagram, а саме рольова модель соціальної мережі, тобто створення постів, коментарів, лайки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна інформація про соціальні мережі

Соціальна мережа – це вебплатформа або мобільний застосунок, призначений для спілкування, обміну інформацією та контентом між користувачами. Соціальні мережі дозволяють користувачам знаходити і додавати нових друзів, створювати та приєднуватися до груп, обмінюватися повідомленнями та зображеннями, а також ділитися різними контентом – починаючи відео та фото до текстів та мемів.

Соціальні мережі стали дуже популярними з появою таких платформ, як Facebook, Twitter та LinkedIn, і сьогодні вони стали значно більш розповсюдженими та різноманітними. На даний момент соціальні мережі використовуються мільярдами людей по всьому світу, забезпечуючи зручний і легкий спосіб спілкування та обміну інформацією.

Соціальні мережі зазвичай забезпечують різні функції, такі як профілі користувачів, друзі, групи, повідомлення, різні типи контенту, новини та інше. Крім того, вони часто дозволяють користувачам взаємодіяти з брендами, продуктами та послугами, що робить їх ефективними маркетинговими інструментами для бізнесів. Наприклад, Facebook є однією з найбільших рекламних платформ у світі, і його прибуток від реклами значно змінюється залежно від року та інших факторів у 2020 році дохід Facebook від реклами склав приблизно 84,2 мільярда доларів США. Важливо враховувати, що прибуток від реклами Facebook може змінюватися залежно від різних факторів, таких як рекламні тренди, конкуренція на ринку, зміни в алгоритмах розкладання новин, а також світові події і тренди, такі як пандемія COVID-19. Також до списку відомих компаній з багатомільйонними прибутками входять Twitter, Instagram, LinkedIn, TikTok та інші.

1.2 Загальна інформація про децентралізовані соцмережі

Децентралізовані соціальні мережі – це новий напрямок у соціальних мережах, який з’явився в останні пару років, та розвивається шаленими темпами, якщо коротко, то це соціальні мережі до яких суспільство звикло, що функціонують на основі блокчейн-технологій, в яких користувачі контролюють свої дані і зберігають їх на своїх власних серверах. Основна ідея децентралізованих соціальних мереж полягає в тому, що вони дозволяють користувачам бути власниками своїх даних та контролювати їх розповсюдження, а не залежати від одного централізованого постачальника [6–9].

Децентралізовані соціальні мережі є відповіддю на проблеми централізованих соціальних мереж, такі як порушення приватності користувачів, контроль над даними користувачів та можливість цензури. Для прикладу розглянемо скандальну ситуацію, яка трапилася у 2018 році, Cambridge Analytica отримала дані про мільйони користувачів від Facebook використовувала ці дані для створення детальних профілів користувачів і налаштування політичної реклами, включаючи підтримку кампанію одного з кандидатів.

У предметній області децентралізованих соціальних мереж важливо враховувати наступні аспекти.

Безпека та приватність даних – це найбільш важливий аспект у децентралізованих соціальних мережах, оскільки користувачі мають повне контролювання над своїми даними. Тому, необхідно забезпечити безпеку та приватність даних за допомогою різноманітних методів, таких як шифрування даних, використання протоколів безпеки та ін.

Масштабованість – децентралізовані соціальні мережі повинні бути масштабовані для забезпечення швидкого та безперебійного функціонування системи, навіть при збільшенні кількості користувачів.

Відкритість та прозорість – децентралізовані соціальні мережі повинні бути відкритими та прозорими для забезпечення вільного обміну інформацією.

1.3 Детальніше про Web3 та Блокчейн

Web3 – це третє покоління Інтернету, яке базується на технології блокчейн та децентралізованій обробці даних. Web3 дозволяє користувачам контролювати свої дані та інтернет-ідентичність, не залежати від посередників, таких як соціальні мережі, та більш безпечно і приватно взаємодіяти з іншими користувачами.

Однією з основних принципів Web3 є децентралізація. Це означає, що замість того, щоб зберігати дані на централізованих серверах, вони зберігаються на різних комп'ютерах, які називаються вузлами. Кожен вузол має копію всіх даних, що забезпечує більш високий рівень безпеки та надійності.

Ще однією важливою складовою Web3 є технологія блокчейн, яка забезпечує безпечну та прозору обробку даних. Блокчейн (англ. blockchain) – це розподілена система збереження даних, що базується на криптографічних принципах та децентралізованому управлінні. Він функціонує як ланцюг блоків, кожний з яких містить набір транзакцій.

Блокчейн не має центральної влади або сервера, що керує ним. Замість цього, він розподілений між багатьма вузлами (комп'ютерами), які спільно управляють блокчейном.

Кожен вузол блокчейну має повну копію всіх даних, що зберігаються в блоках. Це забезпечує стійкість та надійність даних, оскільки відновлення інформації можливе з будь-якого вузла.

Кожен блок має унікальний ідентифікатор, який отримується шляхом застосування хеш-функції до даних блоку. Крім того, дані блоку підписуються криптографічним ключем, що гарантує цілісність та невідповідність даних.

У блокчейні використовується алгоритм консенсусу, який дозволяє вузлам домовлятися про правильність та послідовність транзакцій. Існує кілька різних алгоритмів консенсусу, які використовуються в блокчейні і децентралізованих системах. Нижче описані декілька з них.

Proof-of-Work (PoW). Цей алгоритм використовує обчислювальну потужність для досягнення консенсусу. Майнери розв’язують складні математичні завдання, витрачаючи значну обчислювальну енергію. Перший майнер, який успішно розв’язує завдання, отримує право створити новий блок і отримати винагороду. PoW використовується в Bitcoin і Ethereum.

Proof-of-Stake (PoS). В PoS консенсус досягається шляхом відбору вузлів на основі їх власності або ставок. Вузли, які мають більше монет або ставок, мають більшу ймовірність стати обраною стороною для створення нового блоку і отримати винагороду. PoS зменшує споживання енергії, пов’язане з PoW, і використовується в таких блокчейн-платформах, як Ethereum 2.0 та Cardano.

Delegated Proof-of-Stake (DPoS). У DPoS група вузлів обирає делегатів, які представляють мережу і мають право створювати блоки. Кількість делегатів обмежена, і вони обираються голосуванням власників токенів. DPoS використовується в блокчейні EOS.

Коли блок доданий до блокчейну, він стає незмінним та не може бути змінений без консенсусу мережі. Це забезпечує надійність та недоступність для зміни інформації, що знаходиться у блоках.

Формування блоку в блокчейні може варіюватися залежно від конкретної реалізації блокчейну, але основні кроки процесу виглядають так:

Першим кроком є збір транзакцій, які потрібно включити до нового блоку. Транзакції можуть бути створені користувачами системи та передані учасниками мережі для обробки.

Перед тим, як транзакції будуть включені до блоку, вони перевіряються на валідність. Це включає перевірку цифрових підписів, достатньої наявності коштів або дотримання інших правил, які встановлені блокчейном.

Після перевірки транзакцій формується новий блок. Він включає заголовок блоку, який містить метадані, такі як номер блоку, хеш попереднього блоку, час створення і т. д. Також в блоку включаються валідні транзакції з попереднього кроку.

Після формування блоку виконується обчислення хеша блоку. Це зазвичай виконується шляхом застосування криптографічної хеш-функції до даних блоку. Отриманий хеш ідентифікує блок і гарантує його недоступність для змін без зміни хеша.

Остаточний крок – додавання сформованого блоку до блокчейну. Це включає перевірку, чи валідний отриманий хеш блоку та підтвердження консенсусу, в залежності від використовуваного алгоритму консенсусу.

Ці кроки повторюються для кожного нового блоку, що додається до блокчейну [1–5, 7].

У блокчейні транзакція є основною одиницею обміну даними або операції, яку користувачі виконують в межах мережі. Вона представляє собою запис про передачу або зміну даних в блокчейні.

Транзакція визначає, хто є відправником (надає деякі активи або інформацію) і хто є отримувачем (отримує активи або інформацію).

Транзакція містить дані, які передаються від відправника до отримувача. Ці дані можуть представляти фінансові транзакції, контракти, повідомлення або будь-яку іншу інформацію, яка передається в рамках блокчейну.

Користувачі підписують свої транзакції за допомогою цифрового підпису, що гарантує автентичність та цілісність транзакції.

Транзакції можуть потребувати підтвердження від інших учасників мережі, які перевіряють їх правильність та валідність перед включенням до блокчейну. Це забезпечує надійність та безпеку операцій.

У деяких блокчейнах, зокрема у криптовалютних мережах, транзакції можуть бути пов’язані з винагородою для майнерів або вузлів, які підтверджують та обробляють транзакції.

Транзакції у блокчейні є незмінними та безпечними, оскільки вони підписуються цифровими підписами та включаються до блоків, які потім додаються до ланцюжка.

У транзакціях часто люди чи Dapps надсилають криптовалюти(токени) один одному чи визивають методи смарт контрактів.

Всі види криптовалют умовно можна розділити на три основні типи: біткоїн, альткоїни (серед них стейблкоїни), токени (включаючи DeFi-токени). Кожна з цих груп має свої особливості.

Спочатку розглянемо такі поняття як цифрова валюта, віртуальна валюта та криптовалюта.

Цифрова валюта – електронні гроші з властивостями звичайних (фіатних), які можуть бути регульованими та нерегульованими. Це загальний термін для всіх електронних грошей (грошей у цифровому просторі). До цифрової валюти належать віртуальні валюти та криптовалюти. Цифрові гроші називають також кіберготівкою.

Віртуальна валюта – вид цифрової валюти. Усі віртуальні валюти – цифрові (існують лише в інтернеті), але не всі цифрові валюти – віртуальні.

Криптовалюта – різновид віртуальної валюти, створена за допомогою криптографічних методів та математичних обчислень переважно на базі блокчейну. Це і цифрова, і віртуальна валюта, тому що вона існує в інтернеті та створена за допомогою криптографічних алгоритмів.

Станом на липень 2022 року у світі налічувалося понад 19 тис. криптовалют.

Самі собою біткоїн та альткоїни – це не гроші у звичному розумінні, а складний цифровий продукт зі своїм власним криптокодом та зашифрованим записом. Щоб отримати статус грошей, вони проходять складний механізм трансформації та обробки за допомогою спеціальних технологій.

Біткоїн – це перша криптовалюта, яка досі залишається провідною цифровою монетою за ринковою капіталізацією.

Біткоїн – це глобальна однорангова електронна платіжна система, яка дозволяє сторонам здійснювати угоди без посередників від імені банку чи іншої фінансової організації.

Ця криптовалюта часто розглядається як цифрова альтернатива фіатним валютам та золоту. У біткоїна є низка важливих відмінностей від традиційних паперових грошей, до яких суспільство звикло.

Традиційні (фіатні) гроші друкують центробанки, ФРС та інші державні установи. Іноді вони друкують їх так багато, що це призводить до інфляції. Біткоїн же, навпаки, ресурс вичерпний. Його виробництво (емісія) обмежене програмно, і максимально можлива сума в обігу становить 21000000 BTC.

Біткоїн – це монета, яку користувачі можуть передавати один одному безпосередньо, з електронного гаманця на гаманець. Їм не потрібна допомога банків, платіжних систем (на кшталт SWIFT, Visa) та інших фінансових структур. Система є децентралізованою, а копії блокчейна (історії всіх транзакцій, що здійснювалися) зберігаються у всіх користувачів на пристроях.

Це означає, що аніхто не може заволодіти всією мережею і стати чимось на кшталт центрального банку в ролі монополіста. Це теоретично можливо, але Теоретично, так, але практично, заволодіти мережею Bitcoin дуже малоймовірно і дуже складно.

Мережа Bitcoin є децентралізованою, що означає, що вона складається з великої кількості різних вузлів, розподілених по всьому світу. Кожен вузол має свою копію блокчейну і співпрацює з іншими вузлами для підтвердження транзакцій. Це робить мережу більш стійкою до атак і ускладнює заволодіння нею [1–5].

Bitcoin використовує алгоритм консенсусу Proof-of-Work, що вимагає великої обчислювальної потужності для обробки транзакцій та додавання блоків до ланцюжка. Заволодіння мережею вимагало б володіння значною часткою обчислювальної потужності мережі, що є дуже дорогим і складним завданням.

Рішення щодо напрямку та розвитку мережі Bitcoin приймаються за допомогою розподіленого процесу прийняття рішень, включаючи протоколи консенсусу та покликання до спільної згоди. Це ускладнює можливість централізованого контролю чи зламу мережі.

Блокчейн Bitcoin використовує сильні криптографічні методи, такі як цифрові підписи, для захисту транзакцій і перевірки автентичності. Це робить важким підроблення транзакцій або зміну вже затверджених блоків.

Таким чином, заволодіння мережею Bitcoin потребує величезних ресурсів.

Власне, через біткоїн і з’явився термін «криптовалюта».

1.4 Проблеми блокчейнів

Блокчейн, як технологія, не ідеальна, має свої переваги, але також стикається з деякими проблемами.

Одна з головних проблем блокчейна полягає в його масштабованості. Традиційні блокчейн мережі, такі як Bitcoin та Ethereum, можуть обробляти обмежену кількість транзакцій за секунду, що обмежує їх широкомасштабне використання. Це стає проблемою для ситуацій, коли потрібно обробляти великий обсяг транзакцій швидко та ефективно [2].

У деяких блокчейн мережах, зокрема в публічних криптовалютних мережах, вартість транзакцій може бути високою. Виконання транзакцій вимагає обчислювальних ресурсів та майнерських винагород, що може призвести до високих комісій за операції.

В залежності від алгоритму консенсусу, час підтвердження транзакцій у блокчейні може бути відносно довгим. Наприклад, в алгоритмі Proof-of-Work час блокування нового блоку може займати кілька хвилин або більше. Це може бути неприйнятним для деяких застосувань, де важлива миттєва обробка операцій.

В алгоритмі Proof-of-Work використовується значна обчислювальна потужність, що призводить до великого споживання енергії. Це викликає обговорення щодо екологічної стійкості і сталості блокчейнів, особливо коли йдеться про масштабне використання.

Web3 дозволяє користувачам зберігати свої дані та ідентичність на блокчейні та контролювати їх. Крім того, вони можуть взаємодіяти з іншими користувачами без посередників та довіряти їм свої дані, такі як фінансові дані та особиста інформація.

Web3 також дозволяє розробникам створювати децентралізовані проєкти (DApps), які працюють на блокчейні та не залежать від централізованих серверів. DApps можуть бути використані в різних галузях.

У проєкті будемо використовувати блокчейн Lightning, який створений для миттєвих платежів. Lightning виділяється своїм масштабуванням, та маленькою комісією, тому сприяє використанню для трансферу маленьких платежів, адже немає затрат на майнінг блоків [6, 9].

1.5 Що таке Lightning, як він вирішує проблеми Bitcoin

Блокчейн Lightning – це розширення блокчейну Bitcoin, яке пропонує масштабовані та швидкі мікроплатежі, зменшуючи навантаження на головний блокчейн [7].

Блокчейн Lightning дозволяє здійснювати транзакції офф-чейн, тобто поза головним блокчейном Bitcoin. Це дозволяє досягти набагато швидших часів підтвердження та обробки транзакцій, оскільки вони відбуваються безпосередньо між учасниками, не потребуючи підтвердження в кожному блоку.

Lightning Network дозволяє значно покращити масштабованість мережі Bitcoin, оскільки він дозволяє виконувати тисячі транзакцій за секунду, не навантажуючи головний блокчейн. Це робить його більш придатним для масових прийомів мікроплатежів та повсякденних транзакцій.

Завдяки використанню блокчейну Lightning, транзакції можуть виконуватись з низькими комісіями або навіть безкоштовно. Це робить його особливо привабливим для дрібних та повторюваних платежів, які не виправдовують високих комісій, що часто пов’язані з головним блокчейном.

Використання блокчейну Lightning допомагає зменшити навантаження на головний блокчейн Bitcoin, оскільки багато транзакцій можуть бути оброблені офф-чей. Тобто офф-чейн платежі, це такі транзакції, які відбуваються поза основним блокчейном. Офф-чейн технології дозволяють

виконувати швидкі та ефективні транзакції, які не потребують безпосереднього запису в блокчейн.

Офф-чейн технології, такі як блокчейн Lightning, створюють додатковий шар або протокол, який працює поверх головного блокчейна. Ці технології дозволяють учасникам здійснювати безліч швидких та недорогих транзакцій між собою, передаючи інформацію про ці транзакції безпосередньо між вузлами мережі, а не записуючи їх на головний блокчейн [7].

Офф-чейн транзакції відбуваються безпосередньо між учасниками і можуть бути виконані майже миттєво. Вони не потребують підтвердження в кожному блоку, як у головному блокчейні, що дозволяє значно зменшити час очікування та підвищити пропускну здатність мережі [8].

Офф-чейн технології дозволяють обробляти велику кількість транзакцій поза блокчейном, що розріджує навантаження на головний блокчейн і допомагає вирішувати проблему масштабованості.

Використання офф-чейн транзакцій дозволяє знизити комісії, оскільки вони не потребують запису на головний блокчейн. Це особливо важливо для мікроплатежів. Це робить мережу більш ефективною та швидшою, а також зменшує комісії та час очікування.

Також блокчейн Lightning забезпечує певний рівень приватності для користувачів, але варто зрозуміти, що вона відрізняється від приватності, яку можна знайти в інших блокчейнах.

Учасники мережі Lightning можуть використовувати псевдоніми або адреси, які не пов’язані з їх особистими даними або ідентифікацією. Це забезпечує рівень анонімності при здійсненні транзакцій.

В Lightning Network баланси каналів підтримуються поза блокчейном, що утруднює визначення актуальних балансів користувачів. Це допомагає зберігати приватність, оскільки баланси не є публічно доступними.

Мережа Lightning використовує механізми маршрутизації для передачі платежів від одного вузла до іншого. Ці механізми забезпечують захист приватності, оскільки маршрутизатори не повинні знати деталі платежу, що проходить через їхні вузли.

В Lightning Network транзакції відбуваються офф-чейн, тобто поза головним блокчейном Bitcoin. Це дозволяє зберігати приватність, оскільки деталі транзакцій не реєструються на головному блокчейні.

Важливо враховувати, що хоча блокчейн Lightning надає певний рівень приватності, він не є повністю анонімним. Деяка метаінформація, така як дані маршрутизації і часові позначки, можуть бути доступними, що дозволяє проводити аналіз ефективності мережі та розуміти напрям платежів.

Розглянувши рисунок 1.1, можемо побачити, що Lightning працює таким чином, що користувачі, які є вузлами мережі, відкривають канали один до одного, по який вони відправляють криптовалюту, коли ліміт цього каналу вичерпується він закривається, та усі гроші переходять з off-chain у on-chain, використовуючи усього 2 транзакції, замість *n*-ї кількості.

Саме тому зараз Lightning є дуже привабливим для написання DApps, адже він є швидким, легко маштабується, низькі комісії та має покращену приватність порівняно з Bitcoin.

За останні роки екосистема Lightning Network значно розширилась. З'явилися різноманітні бізнеси, платіжні шлюзи та застосунки, які підтримують використання Lightning для реальних транзакцій. Це допомагає популяризувати та розвивати цю технологію.

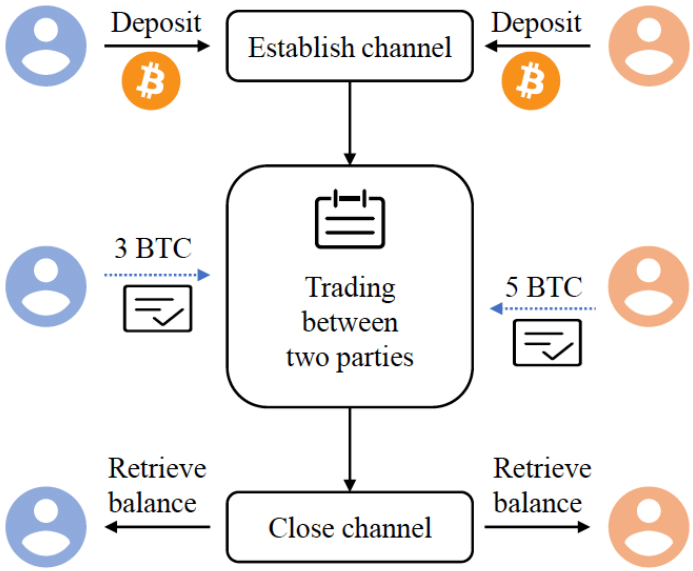

Рисунок 1.1 – Принцип роботи Lightning мережі

З’являються нові розробки та поліпшення у протоколі Lightning Network. Розробники активно працюють над зменшенням складності використання, покращенням безпеки та розширенням функціональності мережі. Це сприяє довірі до технології та її подальшому прийняттю.

1.6 Детальніше про Web5

Останнім часом було декілька концепцій та ініціатив, які пропонували майбутні розширення вебтехнологій після Web3, таких як Web3.5, Web4 і т.д. Кожна з цих концепцій передбачає розвиток інтернету та вебтехнологій у новому напрямку, який може включати покращену децентралізацію, розширені можливості штучного інтелекту, розподілені реєстри, розширену взаємодію з пристроями Інтернету речей та інші інновації [10–15].

Web5 – це децентралізоване бачення вебу, побудоване повністю на блокчейні Bitcoin, ця концепція була представлена співзасновником Twitter Джеком Дорсі. Яке ставить перед собою мету захисту особистих даних, повертаючи власність над цифровою ідентичністю користувачам. Практично, воно використовує мережу Lightning Bitcoin – протокол, який працює поза основним блокчейном, щоб забезпечити швидку функціональність мережі незалежно від токенів, комісій за транзакції, довірених валідаторів чи додаткових механізмів консенсусу – для створення інфраструктури персональних серверів, у якій комп’ютери та інші пристрої підключаються безпосередньо один до одного, утворюючи мережу рівноправних вузлів без необхідності централізованого сервера чи посередника. У такій мережі кожен вузол може виступати як клієнт та сервер одночасно, обмінюючись ресурсами, послугами або даними з іншими вузлами [13–15].

Позиціонуючи користувачів як єдиних керівників своїх даних – закодованих у гаманці, збережених на вузлі або пристрої – вони тепер мають вибір робити особисті дані і інформацію публічними або відкликати доступ до них.

Головна місія Web5 полягає у створенні системи, непроникної для повноважень великих технологій. Для цього Web5 поєднує в собі зручність Web 2.0 із місією децентралізації, започаткованою дизайном Web3 [7, 10–13].

Відповідно до рисунку 1.2 можна зрозуміти, що у Web2 користувачі не володіють своєю інформацією, вона належить централізованим компаніям, а у Web5 уся інформація належить користувачам, а вони в свою чергу установлюють, який сервіс може використовувати їхні дані, а які ні.

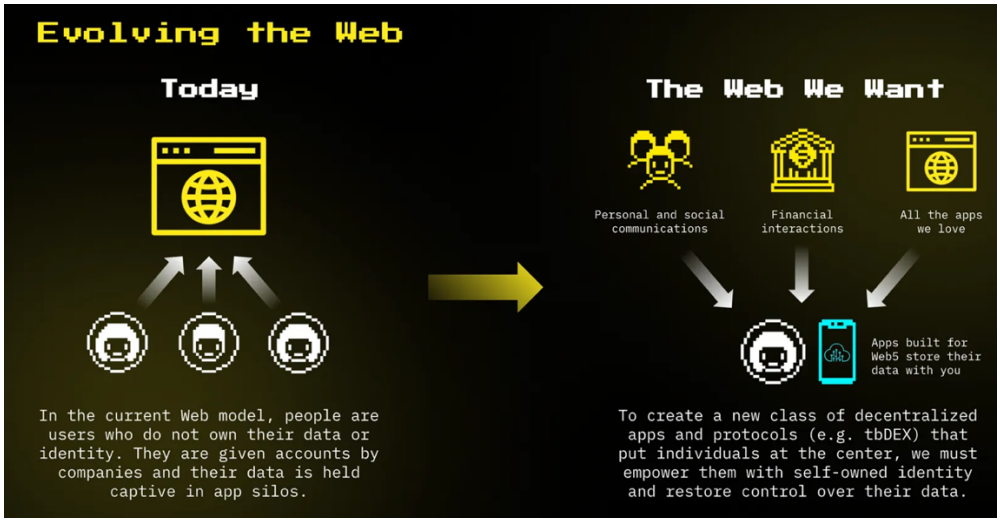

Рисунок 1.2 – Порівняння принципів роботи Web 2.0 та Web 5.0

Основними функціями Web5 є ідентифікатори, які належать користувачам, облікові дані, які можна перевірити, і децентралізовані вебвузли, якими можуть бути домашні комп’ютери чи ноутбуки з високошвидкісним підключенням до Інтернету. Вони служать робочими центрами мережі, відповідальними за зберігання даних, ретрансляцію повідомлень, виконання протоколів і запуск децентралізованим застосункам. Зрештою, dApps запускають програмні протоколи, які дозволяють вузлам та користувачам Web5 взаємодіяти та обмінюватися даними.

Людство вже бачило успішні реалізації P2P системи на прикладах BitTorrent, яка демонструє успішну однорангову систему з відкритим вихідним кодом із службами обміну файлами, Napster, який зробив те саме для музики, хоча й нелегально.

Визначною особливістю Web5 є його нова можливість перенесення децентралізованих цифрових ідентифікацій.

Цифрова ідентичність – це спосіб ідентифікації людини або організації в онлайн середовищі. Вона включає в себе інформацію про особу або організацію, яка дозволяє перевірити її автентичність та дозволяє взаємодіяти з іншими користувачами та сервісами в цифровому просторі.

Цифрова ідентичність повинна бути захищена від несанкціонованого доступу і зламів. Це може бути досягнуто за допомогою різних методів аутентифікації, таких як паролі, біометричні дані та криптографічні ключі.

Цифрова ідентичність повинна захищати приватні дані користувача, включаючи особисту інформацію та історію взаємодії з іншими користувачами та сервісами. Деякі системи цифрової ідентифікації можуть використовувати блокчейн технологію для забезпечення конфіденційності та безпеки даних.

Цифрова ідентичність повинна бути легко переносна між різними сервісами та аплікаціями. Це дозволяє користувачам використовувати одну і ту ж ідентичність для доступу до різних сервісів та застосунків, зменшуючи кількість необхідних реєстрацій та авторизацій.

Цифрова ідентичність дозволяє користувачам взаємодіяти з іншими користувачами та сервісами в цифровому просторі, забезпечуючи надійність та автентичність взаємодії [14].

У кваліфікаційній роботі буде використовуватися DID – це децентралізований ідентифікатор, який дає змогу перевірити децентралізовану цифрову ідентифікацію. DID стосується будь-якого суб’єкта (наприклад, особи, організації, предмета, моделі даних, абстрактної сутності тощо), визначеного контролером DID. На відміну від типових федеративних ідентифікаторів, DID розроблено таким чином, що їх можна відокремити від централізованих реєстрів, постачальників ідентифікаційних даних і центрів сертифікації. Зокрема, хоча інші сторони можуть використовуватися для виявлення інформації, пов’язаної з DID, конструкція дозволяє контролеру DID доводити контроль над ним, не вимагаючи дозволу будь-якої іншої сторони.

DID – це URI, які пов’язують DID коротку версію, яка має такий вигляд *did:ion:EiABlrQV71jvpSL1H711gmxlCwE-H1e8Rojz66xhjPlDnw* із документом DID, який зберігає всю інформацію про ідентифікатор.

Лістинг 1.1 Приклад DID документу:

```
{
  "id": "did:example:123",
  "service": [{
    "id":"#dwn",
    "type": "DecentralizedWebNode",
    "serviceEndpoint": {
    "nodes":["https://dwn.example.com","https://example.org/dwn"]
    }
  }]
}
```

Подібно до глобального унікального імені користувача чи цифрового паспорта, протокол адаптує легку авторизацію входу в різні програми, за винятком того, що інформація для входу – часто збір прізвищ, номерів телефонів і адрес – залишається з користувачем. Це рішення відкидає такі поточні системи, як облікові записи Google або Apple ID, які нескінченно зберігають конфіденційні дані в сховищах великих технологічних конгломератів.

1.7 Web3 vs Web5

Описання Web3 та Web5, знайомо схоже, обидва вони є ітераціями нового покоління Інтернету нового покоління, які працюють на однорангових системах з відкритим кодом і мають спільну мету усунення центральних органів влади та цензури.

«Створення Web5 не означає провал Web3; Це лише два підходи з подібною метою децентралізації Інтернету», – сказав Джош Дрейк, операційний директор Dfinity Foundation, некомерційної організації, яка розробляє загальнодоступний блокчейн, відомий як Internet Computer Protocol. «Головна відмінність полягає в тому, що Web3 представляє різноманітну екосистему розробників із широким спектром існуючих варіантів використання та інновацій, тоді як Web5 є відносно новою платформою, яка зараз знаходиться на концептуальній стадії» [14].

Виявлення різниці – це питання вивчення процесу кожного проєкту, оскільки обидва вебвізуалізації в процесі створення пропонують різну тезу щодо того, як має бути досягнута децентралізація.

1.8 Децентралізоване розподілення даних

Більшість цифрових дій між людьми, організаціями, пристроями та іншими об’єктами вимагає обміну повідомленнями та даними. Щоб об’єкти могли обмінюватися повідомленнями та даними для потоків облікових даних, програм або сервісів, їм потрібен інтерфейс, через який можна було б зберігати, виявляти та отримувати дані, пов’язані з потоками та досвідом, у яких вони беруть участь.

Децентралізований вебвузол (DWN) – це об’єкти механізму зберігання даних і ретрансляції повідомлень можуть використовувати для пошуку загальнодоступних або приватних даних, пов’язаних із заданим децентралізованим ідентифікатором (DID) [16].

Децентралізовані вебвузли – це сітчаста структура сховища даних, яка дозволяє об’єкту керувати декількома вузлами, які синхронізуються в одному стані один з одним, дозволяючи об’єкту-власнику захищати, керувати та обмінюватися своїми даними з іншими, не покладаючись на місцезнаходження чи специфічна інфраструктуру постачальника, інтерфейси або механізми маршрутизації [9–13].

1.9 Огляд існуючих децентралізованих соцмереж

Існують такі децентралізовані соціальні мережі як Diaspora, Mastodon та інші, проаналізувавши бізнес модель та клієнтську частину цих мереж, було створено список того, що потрібно покращити у цих соцмережах.

Інтерфейс значно поступається таким мастодонтам як Instagram, Twitter, а також є дуже перенавантажений деталями, про які користувач може не знати. Але вони більш спрямовані на дуже вузьку групу людей, які є експертами у даній сфері, тобто ці соціальні мережі не можуть бути популярними, адже функціонал на інтерфейс дуже примітивний, а також піар-кампанія цих соціальних мереж не є успішною, а також частина соціальних мереж має Web5 характеристику, а частина Web3, тому мою соціальна мережа може вирішити цю проблему з комбінуванням технологій, а також за мету було поставлено створення соціальної мережі на велику кількість людей, а добрим UI/UX, який є простим, але інформативним та зручним у використанні.

1.10 Постановка задачі

Проаналізувавши існуючі платформи у ніші децентралізованих соціальних та стан сучасних централізованих соціальних мереж, можна наголосити про актуальність проблеми та висунути ряд функціональних і технічних вимог щодо розроблюваного MVP застосунку.

Об’єктом роботи є децентралізований процес передачі даних між користувачами соціальної мережі на основі технологій Web3 та Web5 технологій з використанням DID, DWN.

Метою роботи є побудова соціальної мережі за допомогою сучасних технологій DID та DWN, які допоможуть створити таку систему, що надає людям можливість безпечно зберігати приватну інформацію, яка буде закодована за допомогою децентралізованого ідентифікатора, інформація не

буде проаналізована сервісом за рахунок децентралізованого зберігання, а надає користувачам можливість самим керувати даними.

Одна з найголовніших цілей – створення такої соціальної мережі, що не буде тримати людей у заложників системи, а надасть можливості перенести свої дані та ресурси на свій домен.

Для досягнення мети необхідно вирішити такі завдання:

– визначити функціональність проєкту;

– проєктування архітектури застосунку;

– обрати інструменти за допомогою яких буде створений проєкт;

– створення дизайну;

– розробка застосунку;

– тестування проєту.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Вибір архітектури застосунку

Архітектура проєкту це один з найважливіший аспектів у написані проєктів, адже добре спроєктована система робить підтримка і розширення проєкту більш простими. Модульна структура та відокремлені компоненти дозволяють легко змінювати, додавати або видаляти функціональність без впливу на інші частини системи.

Правильно спроєктована архітектура дозволяє ефективно масштабувати проєкт у коли на це є потреба. Вона дозволяє розподілити навантаження, збільшити продуктивність та забезпечити гнучкість системи.

Якщо проєкт має продуману архітектуру, то підтримка і розширення проєкту стають більш простими. Модульна структура та відокремлені компоненти дозволяють легко змінювати, додавати або видаляти функціональність без впливу на інші частини системи.

Добре спроєктована архітектура забезпечує надійність системи. Вона дозволяє ізолювати проблеми, забезпечує відновлення в разі збоїв та забезпечує високу доступність системи.

Добре спроєктована архітектура сприяє швидкості розробки проєкту. Вона дозволяє розподілити завдання між командами розробників, раціоналізувати процеси розробки та забезпечити високу продуктивність.

Хороша архітектура дозволяє ефективно використовувати і перевикористовувати код. Модульна структура та належне розділення відповідальності дозволяють створювати загальні компоненти, які можуть бути використані в різних частинах проєкту.

Існує кілька основних архітектур для розробки застосунків.

Монолітна архітектура – це традиційна архітектура, в якій весь застосунок розгортається як єдиний компонент. Усі функції і модулі знаходяться в одному великому програмному пакеті. Це простий підхід до

розробки, але може бути складним для масштабування та підтримки великих проєктів.

Клієнт-серверна архітектура – підхід до архітектурі, у якій застосунок розділяється на дві основні частини – клієнтську і серверну. Клієнтська частина взаємодіє з користувачем і надає йому інтерфейс, тоді як серверна частина обробляє бізнес-логіку та зберігає дані. Цей підхід дозволяє розподілити навантаження і полегшує розширення та модифікацію системи [17–21].

Розподілена архітектура – така архітектура, у якій компоненти застосунку розташовані на різних фізичних або логічних вузлах, які працюють разом для досягнення спільної мети. Це може включати розподілені сервери, кластери, мережі, мікросервіси тощо. Розподілена архітектура дозволяє забезпечити високу доступність, масштабованість та надійність системи [17].

Стратегія сервісної орієнтації (SOA) – це підхід до розробки, де застосунок розбивається на набір сервісів, які можуть бути незалежними компонентами з власними інтерфейсами та функціональністю. Ці сервіси можуть взаємодіяти один з одним.

Серед цього списку можна виділити монолітну та розподілену архітектуру.

Але, потрібно проаналізувати недоліки цих систем.

З монолітною архітектурою пов’язана складність розгортання та поновлення системи. Зміни в одній частині коду можуть вплинути на весь застосунок, що потребує повного перегляду та тестування системи перед розгортанням.

У монолітній архітектурі різні компоненти і модулі часто сильно зв’язані між собою. Це означає, що зміни в одному модулі можуть мати непередбачувані наслідки на інші частини системи, що ускладнює розширення та підтримку застосунку.

Монолітна архітектура часто має обмежену масштабованість. Якщо потрібно збільшити пропускну здатність або обробити більше навантаження,

необхідно масштабувати всю систему, навіть якщо тільки окремі компоненти потребують більше ресурсів.

Великі монолітні застосунки можуть бути складними для розвитку і підтримки. З кожним додаванням нових функцій або змінами в системі, збільшується складність розуміння та модифікації коду.

Монолітна архітектура може бути менш гнучкою у порівнянні з іншими архітектурними підходами, такими як мікросервісна архітектура. Зміни в одній частині системи можуть мати вплив на весь застосунок, що ускладнює внесення змін і швидку реакцію на вимоги ринку.

Проаналізувавши існуючі патерни проєктування високонавантажених систем було прийнято рішення використовувати мікросервісну архітектуру.

Мікросервісна архітектура також має свої переваги, але також включає деякі недоліки.

З більшою кількістю сервісів виникає складність управління цими сервісами. Координація і моніторинг багатьох незалежних сервісів може бути викликом. Крім того, зміни в одному сервісі можуть мати вплив на інші, що потребує уважного планування та комунікації.

Мікросервісна архітектура передбачає взаємодію між різними сервісами через мережу. Це може вплинути на продуктивність та завадити швидкість виконання системи, особливо якщо велика кількість сервісів взаємодіє між собою [22, 23].

З розширенням кількості сервісів збільшується складність тестування системи. Необхідно покрити всі сервіси окремими тестами, а також враховувати їх взаємодію. Це може вимагати багато ресурсів та часу для налагодження та валідації системи в цілому.

Використання мікросервісної архітектури може мати вплив на продуктивність системи. Взаємодія між різними сервісами може вимагати збільшеної мережевої пропускної здатності та затримок, особливо якщо вони розгорнуті на різних фізичних серверах або облікових записах.

Порівнюючи обидва підходи мікросервісна архітектура виглядає більш привабливою для застосунку адже проєкт має складну функціональність та передбачається його значне розширення в майбутньому.

Незалежна розробка, розгортання та масштабування окремих компонентів системи є важливими.

Потрібна висока гнучкість та можливість швидкої модифікації окремих сервісів. Існує необхідність в масштабованості та горизонтальному розподілі навантаження.

2.2 Що таке мікросервісна архітектура

Мікросервісна архітектура є підходом до розробки програмного забезпечення, де застосунок будується як набір невеликих автономних компонентів, відомих як мікросервіси. Кожен мікросервіс виконує окрему функцію та має своє власне обмежене контекстне середовище. Ці мікросервіси взаємодіють один з одним через легковагові механізми комунікації, такі як мережеві протоколи або спільні бази даних [24].

Основна ідея мікросервісної архітектури полягає в розбитті складного програмного проєкту на невеликі, незалежні компоненти, які можуть бути розгорнуті та масштабовані окремо. Кожен мікросервіс може бути розроблений, тестований, випущений та масштабований незалежно від інших, що дозволяє командам розробників працювати паралельно та швидко вносити зміни без впливу на інші компоненти системи.

Мікросервіси можуть бути написані різними мовами програмування та використовувати різні технології, що дозволяє використовувати найкращі знаряддя для кожної конкретної задачі. Кожен мікросервіс може мати свою власну базу даних або використовувати спільну базу даних з іншими мікросервісами.

Мікросервіси взаємодіють між собою через інтерфейси програмування застосунків (API), які можуть бути синхронними чи асинхронними. Це

дозволяє мікросервісам комунікувати та обмінюватися даними, незалежно від мови програмування або технології, в якій вони реалізовані [25].

Підсумувавши порівняно з монолітами мікросервісна архітектура надає багато можливостей для гнучкої розробки та підтримки великих проєктів.

2.3 Вибір інструментів для створення проєкту

Визначившись з патерном архітектуру необхідно вибрати інструменти за допомогою якого буде створено застосунок. Почнемо з мови програмування.

Проаналізувавши список сучасних мов було обрано мову Golang. Нижче описані плюси цієї мови завдяки яким вона ідеально вписується в архітектуру проєкту.

Go пропонує простий та лаконічний синтаксис, що полегшує розробку. Вона також відома своєю швидкодією та ефективним використанням ресурсів, що дозволяє побудувати швидкі та ефективні мікросервіси [17].

Однією з сильних сторінок Go є його вбудована підтримка конкурентності. Go має горутини (goroutines) та канали (channels), що дозволяють легко створювати конкурентні та паралельні операції. Це особливо корисно в мікросервісній архітектурі, де багато компонентів можуть працювати паралельно.

Go має багато вбудованих пакетів для роботи з мережевими операціями, такими як HTTP, RPC та інші. Це дозволяє легко створювати та взаємодіяти з мережевими мікросервісами.

Go відома своєю високою надійністю та стійкістю до помилок. Вона надає механізми для ефективного керування помилками та панування над ними, що дозволяє створювати стійкі до збоїв мікросервіси.

Go компілюється в один самостійний виконуваний файл, що полегшує розгортання та виконання мікросервісів. Не потрібні додаткові залежності або встановлення рантайму на сервері.

Для мікросервісної архітектури буде використовуватися такой протокол, як protobuf, який дозволяє описувати API шар, який відповідає за комунікацію між мікросервісами, а також який виконує роль контролеру, тобто проміжним шаром між Model та View, тобто back-end та front-end`ом [24].

За допомогою синтаксису proto3, були описані усі ендпоінти з об’єктами, які вони приймають та віддають.

2.4 Принципи чистого коду

Написання чистого коду – це важлива складова процесу розробки програмного забезпечення. Чистий код – це код, який легко зрозуміти та зберігати, що забезпечує його легку підтримку та розвиток. Ось декілька принципів, які можна використовувати при написанні чистого коду.

Код повинен бути простим та зрозумілим. Це допомагає забезпечити його легку підтримку та розуміння.

Кодова база повинна бути добре задокументованим. Короткі пояснення функцій та змінних допоможуть розуміти код іншим розробникам, які працюватимуть з ним у майбутньому.

Код повинен бути спрощеним та не містити зайвих складнощів. Це допомагає забезпечити його легку підтримку та зміну в майбутньому.

Кодова база повинна бути розділеним на модулі та класи, що дозволяє зменшити залежності між різними частинами програмного забезпечення. Це допомагає забезпечити більшу гнучкість та зручність у підтримці коду.

Окремі модулі повинні бути написаний таким чином, щоб його можна було використовувати у різних частинах проєкту або навіть у різних проєктах. Це допомагає зменшити затрати на розробку та забезпечити більшу ефективність [25–28].

Код повинен бути написаний таким чином, щоб його можна було легко тестувати та перевіряти на наявність помилок. Це допомагає забезпечити

високу якість програмного забезпечення та зменшити ризики виникнення помилок у майбутньому.

Загалом, усі ці правила описані у принципах SOLID.

Принципи SOLID – це набір принципів, які допомагають проєктувати та розробляти програмне забезпечення з високою гнучкістю, розширюваністю та підтримуваністю. SOLID є акронімом, що складається з перших літер кожного принципу. Нижче описані кожний з цих принципів.

Принцип одиничної відповідальності. Кожен клас або модуль повинен мати лише одну причину для зміни. Це означає, що кожен елемент програми повинен бути відповідальним лише за одну конкретну функцію або задачу. Це спрощує розуміння, тестування та зміну коду.

Принцип відкритості/закритості. Система повинна бути відкритою для розширення, але закритою для змін. Це означає, що зміни повинні відбуватись шляхом додавання нового коду, а не зміни існуючого. Це сприяє зменшенню впливу змін на вже працюючий код і забезпечує його стабільність.

Принцип підстановки Лісков. Об’єкти в програмі повинні бути замінюваними своїми похідними типами без зміни коректності програми. Це означає, що класи-нащадки повинні зберігати контракт баового класу, щоб можна було використовувати їх в коді без зміни його логіки.

Принцип інтерфейсу видимості. Клієнти не повинні залежати від інтерфейсів, які вони не використовують. Замість цього, слід розділити інтерфейси на більш малі, спеціалізовані та спрямовані на конкретні функціональні цілі.

Принцип інверсії залежностей. За цим принципом високорівневі модулі не повинні залежати від низькорівневих. І ті, і ті мають залежати від абстракцій. Та абстракції не мають залежати від деталей реалізації. Деталі реалізації повинні залежати від абстракцій.

Це природно, що з розвитком системи зростає її складність. Важливо завжди тримати цю складність під контролем. Інакше може виникнути ситуація, коли додавання нового функціоналу, навіть не дуже складного, обійдеться занадто дорого. Деякі проблеми повторюються особливо часто.

Щоб їх уникати, було розроблено принципи проєктування. Якщо їх дотримуватися, то не буде допущено лавиноподібного підвищення складності системи. Найпростішими такими принципами є SOLID.

2.5 Генерація документації для проєкту

Усі проєктів потребують документації, адже це джерело правди для теперішніх та майбутніх розробників. Для реалізації цієї ідеї було вирішено використовувати Swagger.

Swagger – це набір стандартів та інструментів для опису, створення, документування та використання вебсервісів RESTful API. Swagger дозволяє розробникам та командам зрозуміти, взаємодіяти та спілкуватися з різними сервісами API без необхідності вивчення вихідних деталей або безпосередньої комунікації з розробниками.

Swagger надає можливість автоматично згенерувати документацію для API на основі опису, що спрощує розуміння функціональності та використання API. Це допомагає командам розробників та споживачам API швидко орієнтуватися в можливостях та використанні API.

Swagger дозволяє автоматично згенерувати клієнтський або серверний код для взаємодії з API на основі опису. Це значно спрощує розробку клієнтських програм або серверів, оскільки не потрібно вручну писати та підтримувати весь код для взаємодії з API [22].

Swagger дозволяє виконувати валідацію запитів та відповідей API на основі опису. Це допомагає виявляти та виправляти помилки в API та забезпечує відповідність між документацією та реальною реалізацією API.

Swagger надає можливість створювати інтерактивну документацію, що дозволяє розробникам та споживачам API пробувати та експериментувати з API. На рисунку 2.1 представлено документацію для застосунку, яка була згенерована за допомогою спеціальної команди *protoc*, а також для UI каркасу був використаний темплейт з офіційного сайту Swagger

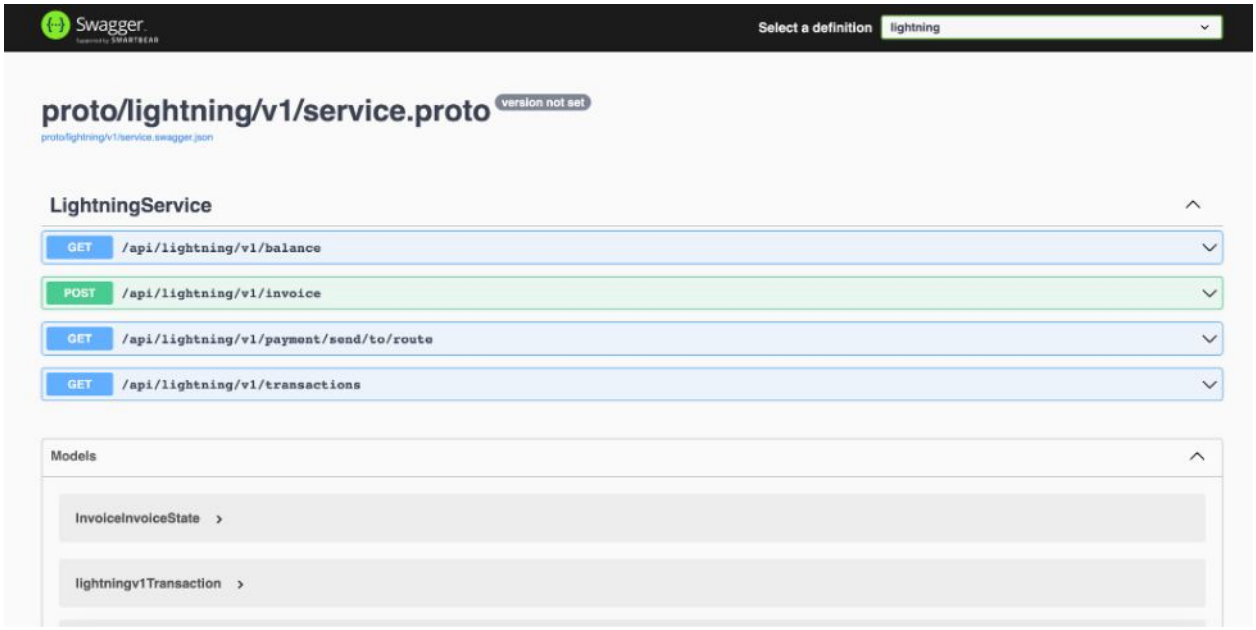

Рисунок 2.1 – Приклад згенерованої документації для сервісу

2.6 Вибір бази даних для проєкту

Вибір бази даних є одним з найважливіших питань, адже у суспільстві абсолютна більшість використовує соціальні мережі кожен день посилаючи десятки тисяч байт даних до серверу, який їх обробляє та зберігає їх у базах даних. Саме тому наша соціальна мережа потребує надійну та стійку до відмов СУБД, яку з легкістю можна буде розширювати, та яка буде підтримувати безліч типів даних та буде мати високу продуктивність. Саме такою БД є Postgres, який вже проявив себе, як одну з найкращих SQL СУБД.

Але дана СУБД має свої недоліки, до яких входять складність налаштування, що може бути складним на перших етапах. PostgreSQL вимагає значного обсягу оперативної пам’яті для ефективної роботи. В порівнянні з деякими іншими СУБД, PostgreSQL може мати обмеження щодо масштабування. Хоча він підтримує реплікацію та кластеризацію, виникнуть труднощі при розподіленому масштабуванні на декілька вузлів, саме ця проблема є однією з найголовніших, адже соціальні мережі мають велике навантаження. А також PostgreSQL використовує багато ресурсів, отже у хмарних сховищах, таких як AWS потрібно створювати досить обширне сховище для нього [21].

Для того, щоб мікросервіси залишалися незалежними та мали можливість для гарного масштабування будемо використовувати принцип за яким кожен сервіс буде використовувати окрему базу даних.

2.7 Огляд архітектури проєкту.

Обравши інструменти для системи перейдемо до проєктування системи. У проєкті були виділення основні компоненти:

- Profile Service – сервіс, який відповідає за створення профайлу для юзерів, та основні дії з ними, так звані CRUD операції;
- DID Service – сервіс, який відповідає за реєстрацію нового юзера, його логіну, та створення DID у офіційному вузлі від Microsoft;
- Lightning Service – сервіс, який відповідає за платежі;
- Lightning Watcher – сервіс, який моніторить всі транзакції, що надходять до системи;
- Comments Service – сервіс, який відповідає за коментарі, та усі операції з ними;
- Posts Service – сервіс, який відповідає за пости, та усі операції з ними;
- Followers Service – сервіс, який відповідає за підписників юзерів, та усі операції з ними;
- Likes Service – сервіс, який відповідає за лайки.

А також було виділено 2 компонента, які відповідають за комунікацію з front-end’ом сайту:

- HTTP Gateway – сервіс, який відповідає за обробку запитів за допомогою протоколу HTTP, у реалізації REST;
- DWN Gateway – сервіс, який взаємодіє з DWN вузлом, та фактично є агрегатором децентралізованого сховища даних.

Усі сервіси було розбито на 2 групи, ті які використовуються HTTP Gateway та ті, що використовуються DWN Gateway.

До групи, яка відноситься до DWN Gateway належать Comments, Posts, Likes, Followers. Саме ці сервіси відносяться до тих, що використовують дані, які належать напряму юзеру, інші сервіси є кастомною реалізацією проєкту, яка використовує конфіденційні дані, наприклад, баланс. Візуальне представлення архітектури проєкту показано на рисунку 2.2, тобто усі запити до серверу проходять через Gateway, і тільки потім потрапляють до сервісу, де вони обробляються.

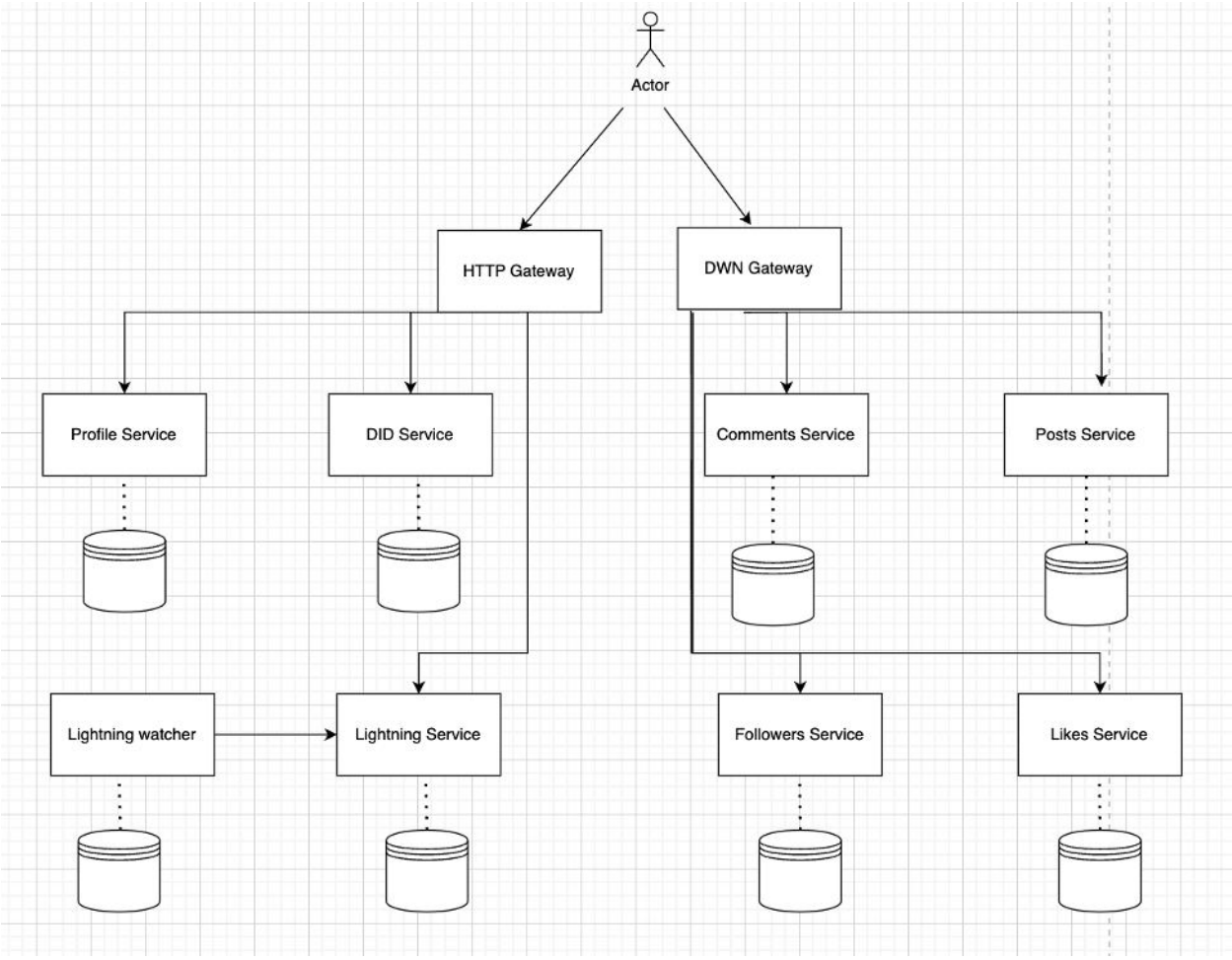

Рисунок 2.2 – Архітектура застосунку

2.8 Аналіз архітектури проєкту з балансуванням навантаження

Так як соціальні мережі мають велике навантаження, нам потрібно як керувати кількома репліками кожного сервісу, щоб витримуватм її. Для проміжної частини, яка вирішує, який сервіс буде виконувати операцію був

обран Nginx, який є вебсервером з відкритим вихідним кодом, який також може виконувати функції проксі-сервера, зворотного проксі, балансувальника навантаження та HTTP-кешування. Він розроблений з урахуванням високої продуктивності та ефективності роботи з багатокористувацькими вебзастосунками.

Nginx розроблений для обробки великого обсягу запитів та забезпечення швидкої передачі даних. Він володіє високою ефективністю, низькою використанням пам’яті та може працювати з багатоядерними системами.

Nginx може виступати в якості проксі-сервера, який пересилає запити до інших серверів, тим самим розподіляючи трафік та покращуючи безпеку. Він також може працювати як зворотний проксі, що приймає запити від клієнтів та передає їх до відповідних серверів [27].

Nginx може використовуватися для розподілу навантаження між декількома серверами, що дозволяє розподіляти запити та забезпечувати високу доступність та масштабування системи.

Nginx підтримує HTTP-кешування, що дозволяє зберігати статичні або рідко змінювані ресурси на сервері. Це сприяє прискоренню відповіді сервера та зменшенню навантаження на вебзастосунок. А також простоту конфігурації та розширення.

Використовуючи плюси Nginx, архітектура буде виглядати, на прикладі Lightning Service та HTTP Gateway. Дана архітектура показана на рисунку 2.3.

Nginx має досить обширний вибір алгоритмів для балансування.

Round Robin – це найпростіший спосіб балансування, де кожен запит прямує до наступного сервера в послідовності. Кожен сервер отримує рівну частину навантаження, що дозволяє рівномірно розподіляти трафік між ними.

Least Connections – спосіб за яким балансування запитів направляються на сервер з найменшою кількістю активних з’єднань. Це дозволяє розподіляти трафік на основі навантаження серверів, забезпечуючи більш ефективне використання ресурсів.

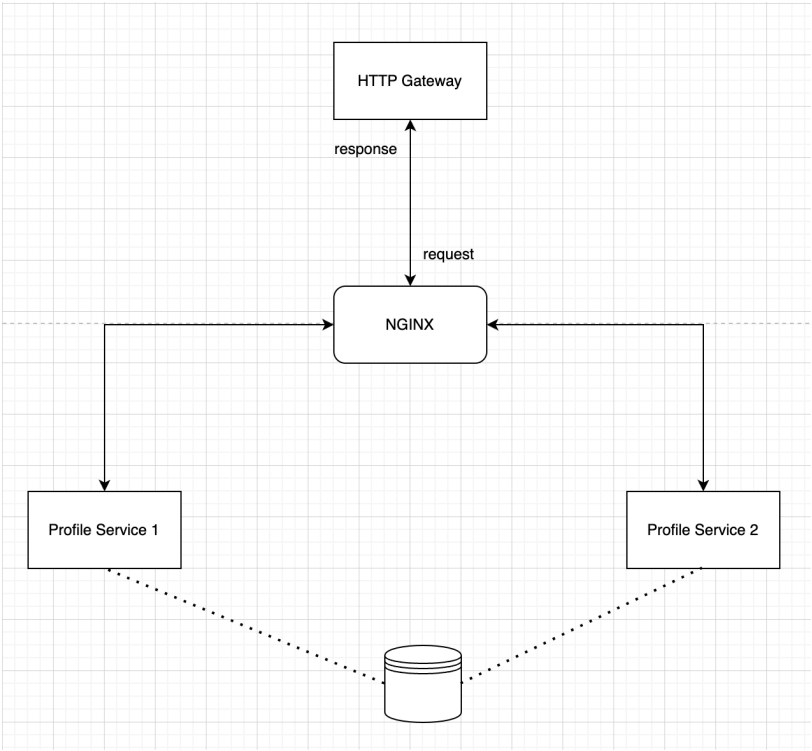

Рисунок 2.3 – Приклад використання Nginx у Lightning сервісі

IP Hash – у цьому режимі балансування Nginx використовує хеш-функцію IP-адреси клієнта для визначення сервера, до якого слід направити запит. Це забезпечує послідовність направлення запитів від одного клієнта до одного сервера, що корисно, наприклад, для сеансових застосунків.

Least Time – Цей спосіб балансування вибирає сервер з найменшим середнім часом відповіді на запити. Він враховує швидкість відповіді серверів для визначення, куди направити наступний запит, що допомагає мінімізувати час відповіді для клієнтів.

Sticky Sessions – цей підхід забезпечує прикріплення клієнта до одного конкретного сервера на певний період часу. Кожен клієнт буде направлений до того ж сервера під час цього періоду, що корисно для застосунків, які потребують збереження.

У проєкті будемо використовувати алгоритм Round Robin, адже він є одним з найпростіших, та потрібно зрозуміти, який сервіс буде мати найбільшу нагрузки, а для цього потрібно мати досить багато юзерів.

2.9 Огляд архітектури для Lightning Service

Lightning Network має гарні параметри для масштабування, але для того щоб не навантажувати один з наших вузлів у проєкті буде застосовано кластер вузлів, який буде балансувати між декількома вузлами Архітектуру кластеру зображено на рисунку 2.4.

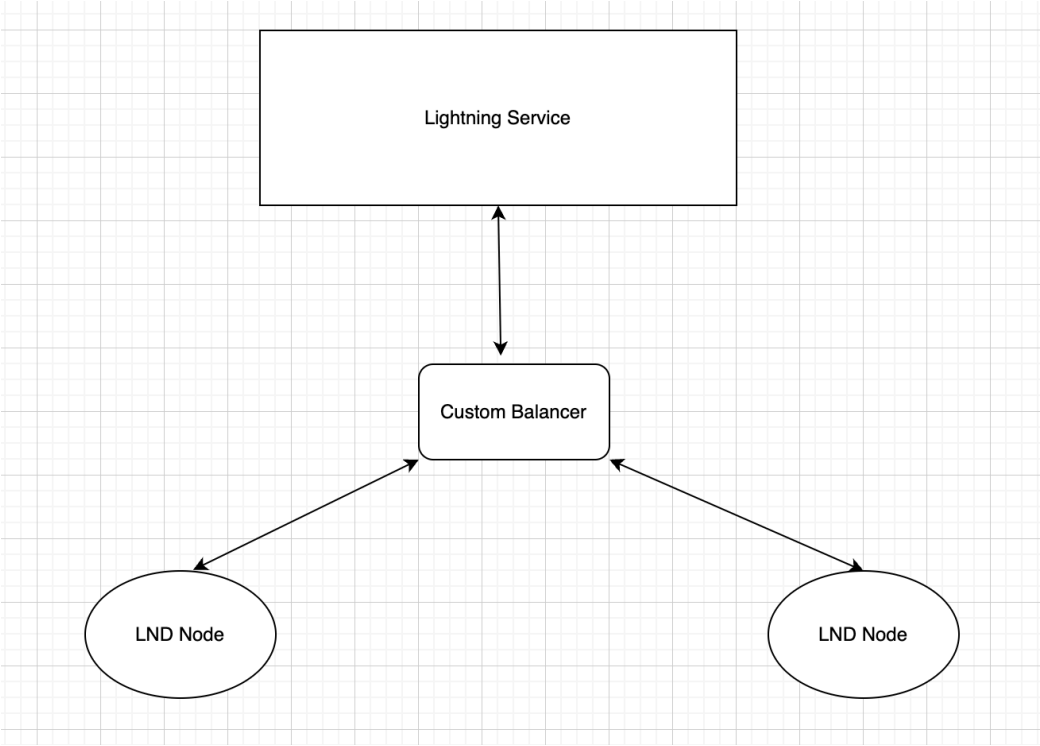

Рисунок 2.4 – Архітектура LND Cluster у Lightning сервісі

Load-balancer буде обирати вузол на який потрібно виконати деяку операцію, а потім повертати результат до місця виклику.

Щоб ініціалізувати кластер Lightning сервіс має визвати метод *New,* який знаходиться у пакеті *lndcluster.*

Лістинг 2.1 Реалізація конструктора для кластеру:

```
// New is Cluster constructor.
func New(configs []client.Config) (*Cluster, error) {
  cluster := &Cluster{
    nodes:          make(map[string]lightning.Interface, len(configs)),
    clusterPubKeys: make([]string, 0, len(configs)),
  }
```

```
  for _, config := range configs {
    node, err := client.New(config)
    if err != nil {
      return cluster, err
    }

    resp, err := node.GetInfo(context.Background())
    if err != nil {
      return cluster, err
    }

    cluster.nodes[resp.IdentityPubkey] = node
    cluster.clusterPubKeys = append(cluster.clusterPubKeys,
resp.IdentityPubkey)
  }

  return cluster, nil
}
```

У цій функції потрібно послідовно пройтись по усім полям кофігур, який сервіс передає у функцію, тобто LND Cluster є гибким сервісом, який може працювати як з одним вузлом, так і з багатьма [7].

Лістинг 2.2 Приклад створення інвойсу у реалізації кластера:

```
func (c *Cluster) AddInvoice(ctx context.Context, in *lnrpc.Invoice)
(*lnrpc.AddInvoiceResponse, error) {
  var (
    inv *lnrpc.AddInvoiceResponse
    err error
  )
  for _, pubKey := range c.shuffledNodes() {
    node, ok := c.nodes[pubKey]
    if !ok {
      c.synchronize()
      continue
    }

    inv, err = node.AddInvoice(ctx, in)
    if err == nil {
      log.Printf("LND CLUSTER: add invoice on node with pub_key %v",
pubKey)
      return inv, nil
```

```
    }
  }

  return inv, err
}
```

Розглядаючи наведений приклад, для генерації інвойсу, можемо побачити, що кластер намагається згенерувати інвойс послідовно на кожній ноді, яку віддає метод *shuffledNodes* до тих пір поки вузол виконає цей виклик без помилки.

Лістинг 2.3 Реалізація перемішування вузлів у кластері:

```
func (c *Cluster) shuffledNodes() []string {
  rand.Seed(time.Now().UnixNano())

  rand.Shuffle(len(c.clusterPubKeys), func(i, j int) {
    c.clusterPubKeys[i], c.clusterPubKeys[j] = c.clusterPubKeys[j],
c.clusterPubKeys[i]
  })

  return c.clusterPubKeys
}
```

Але така кластерізація вузлів має свої проблеми, адже усі платежі, які надходять до отримувача. Для кожного вузла потрібно створювати сервіс, який відповідає за моніторинг вузлів, та усіх платежів, які надходять до системи.

Лістинг 2.4 Реалізація підписки для отримання усіх подій від вузла:

```
invoiceStream, err := s.lightningClient.SubscribeInvoices(ctx,
&lnrpc.InvoiceSubscription{})
if err != nil {
  cancel()
  return errors.Wrap(err, "could not subscribe to invoices")
}

g.Add(func() error {
  for {
    log.Infof("Waiting for invoices from stream...")
```

```
        invoice, err := invoiceStream.Recv()
        if err != nil {
          return errors.Wrap(err, "could not receive invoice")
        }
        log.Infof("Invoice is valid – processing...")

        if s.isClosed.Load() {
          return errors.New("invoice channel closed")
        }

        log.Infof("Sending invoice to the channel...")
        s.invoiceCh <- invoice
        log.Infof("Invoice is sent to the channel")
      }
    }, func(err error) {
      log.Infof("Stop monitoring invoices %v", err)
      cancel()
    })
```

Для отримання повідомлень про оплату інвойсів потрібно створити підписку на усі події пов’язані з платежами [7]. А обробка платежів знаходиться у Lightning сервісі. Для кращого розуміння розглянемо архітектуру цього сервісу, яка показана на рисунку 2.5.

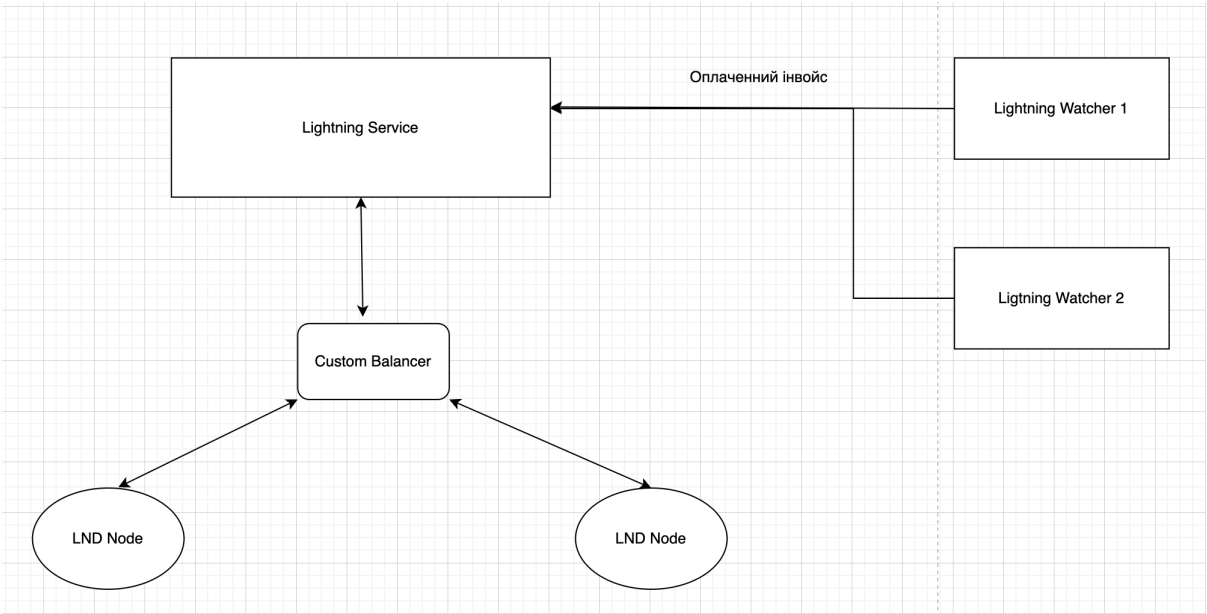

Рисунок 2.5 – Архітектура Lightning Service

Lightning сервіс у свою чергу помічає інвойс як сплачений, який передав один з підписників вузла та збільшує баланс отримувача.

2.10 Контейнерізація проєкту

Контейнеризація – це складання вашої програми у такий незалежний контейнер. Це метод віртуалізації на рівні ОС, що дозволяє розгортати та запускати розподілені застосунки без необхідності запуску цілої віртуальної машини для кожного з них. Тобто декілька ізольованих програм або служб можуть працювати на одному хості та отримувати доступ до одного і того ж ядра операційної системи.

Контейнери застосунків включають файли, змінні середовища, бібліотеки та інші компоненти, необхідні для запуску ПЗ. Вони зазвичай споживають менше ресурсів порівняно із розгортанням на віртуальних машинах. Це тому, що вони не потребують повноцінної операційної системи для підтримки кожного середовища.

Повний набір інформації для виконання в контейнері називається образом. Ці образи на хостах розгортають механізми контейнерів.

Контейнери забезпечують стандартизоване середовище, що дозволяє розробляти та запускати застосунки на різних платформах без необхідності внесення змін. Це дозволяє швидко та ефективно переносити застосунки між різними середовищами, такими як розробка, тестування та продакшн.

Контейнери ізолюють програми одна від одної та від хост-системи, що дозволяє запускати багато контейнерів на одному хості без взаємних впливів. Це допомагає уникнути конфліктів між різними компонентами застосунків та забезпечує надійність та стабільність роботи системи [21, 25].

Контейнери дозволяють швидко розгортати застосунки та їх залежності без необхідності встановлення та конфігурування повноцінних віртуальних машин. Вони можуть бути запущені лише за декілька секунд, що значно прискорює процес розгортання та розвитку застосунків.

Контейнери дозволяють легко масштабувати застосунки, як горизонтально (запуск декількох копій контейнера на різних хостах), так і вертикально (збільшення ресурсів контейнера на одному хості). Це дозволяє відповідати на зростаючі потреби у продуктивності та навантаженні без перерв у роботі системи.

Контейнери дозволяють зберігати та управляти залежностями застосунків разом з кодом. Це дозволяє легко встановлювати та оновлювати необхідні компоненти.

Визначення базового образу можна у Docker створюється за допогомою Dockerfile [28–30].

Dockerfile – це текстовий файл, який містить інструкції для автоматизованої побудови Docker-образу. Dockerfile описує кроки, необхідні для налаштування та розгортання контейнера.

Dockerfile визначає, з якого базового образу буде побудований контейнер. Базовий образ може містити операційну систему, програмне забезпечення та інші компоненти, необхідні для виконання застосунку.

За допомогою Dockerfile можна вказати, які пакети та залежності потрібно встановити у контейнері. Це включає в себе встановлення програм, бібліотек, середовищ виконання та інших компонентів, які потрібні для правильної роботи застосунку.

Dockerfile дозволяє копіювати файли з локальної файлової системи в контейнер. Це може включати копіювання вихідного коду застосунку, конфігураційних файлів, статичних ресурсів тощо.

Dockerfile дозволяє налаштувати середовище виконання контейнера, таке як встановлення змінних середовища, виконання команд, налаштування мережі та портів.

Dockerfile може містити команди, які виконуються під час побудови образу, такі як компіляція коду, налаштування бази даних або інші додаткові операції [23, 26].

Після написання Dockerfile можна використовувати команду `docker build`, щоб автоматично побудувати Docker-образ згідно з інструкціями,

зазначеними у файлі. Dockerfile є потужним інструментом для створення стандартизованих та автоматизованих процесів розгортання контейнерів.

Для застосунку можна використовувати такий Dockerfile.

Лістинг 2.5 Реалізація Dockerfile для застосунку:

```
FROM public.ecr.aws/fivi-stack/golang:1.18.4 AS builder
ADD . /go/src/github.com/bohdanserdonov/fivi
WORKDIR /go/src/github.com/bohdanserdonov/fivi
ARG MOCKS=false
RUN mage services $MOCKS
RUN mage -compile ./build/mage

FROM alpine:3
RUN apk add --no-cache libc6-compat
WORKDIR /root/fivi/
COPY --from=builder /go/src/github.com/bohdanserdonov/fivi/build/* ./
COPY --from=builder /go/src/github.com/bohdanserdonov/fivi/services
./services/
COPY --from=builder /go/src/github.com/bohdanserdonov/fivi/docs ./docs/
ENV PATH="/root/fivi:${PATH}"
```

Було створено команду *mage services*, яка створює бінарні файли для кожного сервісу, а після цього ці бінарні файли разом з папкою з документацією додається до контейнеру [25].

Застосунок буде складатися із багатьох контейнерів, які будуть створені за допомогою Docker Compose.

Docker Compose – це інструмент, який дозволяє описувати та управляти багатоконтейнерними застосунками з використанням одного файлу конфігурації. Він дозволяє визначити компоненти застосунку, налаштувати їх параметри та взаємодію між ними.

Основна мета Docker Compose – спростити процес розгортання та керування складними проєктами, які складаються з кількох контейнерів. За допомогою Docker Compose можна описати весь стек застосунок у одному файлі YAML, що робить його легко зберігати та управляти.

Можна визначити всі компоненти застосунок, включаючи контейнери, мережі, об’єкти збереження даних та інші ресурси, у файлі docker-compose.yml. Це дозволяє легко створювати та налаштовувати сервіси.

Docker Compose дозволяє однією командою запустити або зупинити всі компоненти застосунок. Можна вказати кількість реплік кожного сервісу, налаштувати мережі, прив’язки портів та інші параметри.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОЄКТУ

При розробці застосунку його було розбито на основні частини. Такі як сторінка реєстрації, домашня сторінка, сторінка створення публікації, сторінка профілю, а також функціонал для написання коментарів, редагування профілю тощо.

3.1 Розробка сторінки реєстрації

Найперше, що бачить користувач коли заходить на сайт – це авторизація. Зазвичай соцмережі користуються звичайною схемою, коли вимагають від юзера email чи номер телефону, а також пароль. У застосунку буде використовуватись mnemonic фраза. Mnemonic фраза, також відома як seed-фраза або recovery phrase, є послідовністю слів, яка використовується для відновлення або створення криптовалютного гаманця (wallet). Вона генерується за допомогою спеціального алгоритму і містить 12, 18 або 24 слова.

За допомогою mnemonic фрази можна відновити криптовалютний гаманець у випадку втрати або пошкодження. При створенні гаманця генерується випадкова mnemonic фраза, яка слугує ключем до відновлення гаманця та доступу до збережених у ньому криптовалютних активів.

Mnemonic фраза використовується для створення приватних ключів, які є основою безпеки і доступу до криптовалютних активів. За допомогою спеціального алгоритму mnemonic фраза перетворюється на послідовність приватних ключів, які використовуються для підпису транзакцій та контролю доступу до активів [27].

Mnemonic фраза створюється таким чином, щоб бути легкою для запам'ятовування людиною. Вона складається зі словникових слів, які можна легко відтворити у правильному порядку. Це полегшує використання та зберігання гаманця, оскільки необхідно запам'ятати лише послідовність слів.

Mnemonic фраза є важливим елементом безпеки криптовалютного гаманця. Вона повинна бути добре захищена і не повинна ніколи надсилатися або зберігатися в електронному вигляді, що може бути доступним для зловмисників.

Саме через ці переваги у кваліфікаційній роботі буде використовуватися mnemonic фраза, а також парою до неї буде вказуватися email, для того, щоб користувачі могли безпечно, адже існує можливість підібрати цю фразу та отримати доступ до гаманця чи акаунта іншого користувача, саме тому, юзер при авторизації буде вказувати email.

На рисунку 3.1 можемо побачити, як виглядає найперша сторінка, на яку потрапляє потенційний користувач. Після реєстрації буде згенеровано авторизаційний токен JWT.

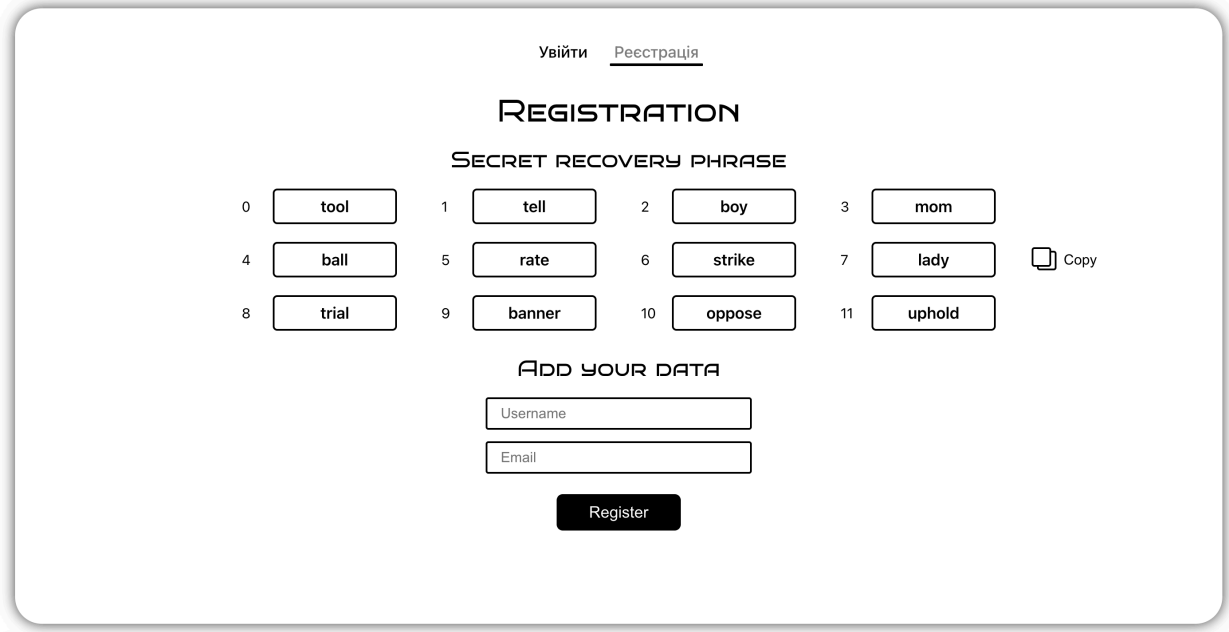

Рисунок 3.1 – Екран реєстрації нового користувача

JWT використовується для аутентифікації та авторизації користувачів. Після успішної аутентифікації сервер генерує JWT, який надсилається до клієнта. Клієнт зберігає JWT і надсилає його у заголовку або запиті до сервера при кожному наступному запиті. Сервер перевіряє підпис JWT і корисне навантаження, щоб підтвердити автентичність і авторизацію клієнта.

JWT має декілька переваг, таких як простота використання, можливість передавати корисну інформацію в зашифрованому форматі і без необхідності зберігання стану на сервері.

А також разом з JWT генерується пара приватного та публічного ключа, яка буде використовуватися для підпису даних, які будуть послані до DWN вузла.

Якщо користувач вже є зареєстрований у системі, йому потрібна сторінка для логіну, яка має вигляд, який показано на рисунку 3.2.

Рисунок 3.2 – Сторінка логіну користувача

На рисунку 3.2 можемо побачити, що потрібно у текстове поле з підказкою Add secret phrase потрібно ввести mnemonic фразу, яку юзер має записати на момент реєстрації, саме ця seed фраза відповідає за роль паролю, Та повинна бути записана у безпечне місце, часто люди записують її на аркуш, але цей кдаптик паперу не потрібно загубити.

3.2 Домашня сторінка користувача

Після реєстрації чи логіну користувачі переходять всередину застосунку. Їх зустрічає екран, який показано на рисунку 3.3.

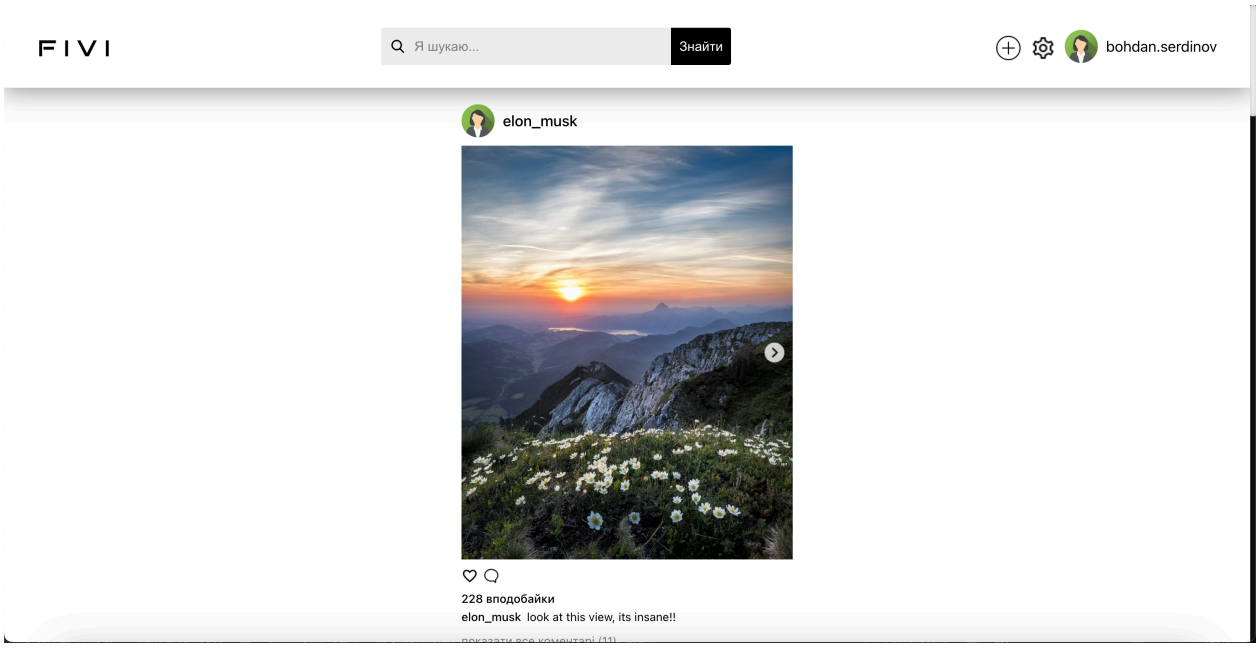

Рисунок 3.3 – Домашня сторінка користувача

На рисунку 3.3, інтерфейс нагадує дизайн соцмережі Instagram, але є один нюанс, що усі дані зберігаються у DWN.

Розглянемо функціонал, який знаходиться на хедері сайту. На шапці сайту можемо побачити функціонал за допомогою якого можна створити пост, можна знайти користувачів за допомогою пошуку, а також можна перейти до сторінки користувача та його налаштування.

Розглянемо модальне вікно для створення посту, яке показано на рисунку 3.4. На даному модальному вікні можемо побачити, що користувачі можуть обрати фотографії та написати для них опис. Після того, як юзер натисне на кнопку Create буде сформований DWN message.

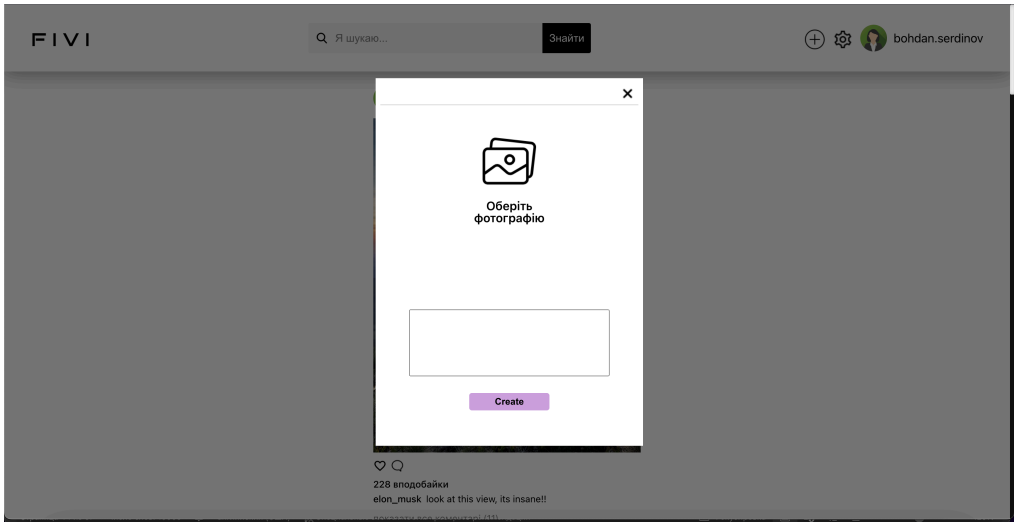

Рисунок 3.4 – Модальне вікно створення посту

3.3 Публікація посту до DWN вузлів

DWN Message, який описано у лістингу 3.1 підписується за допомогою DID Private Key, формується запит, та надсилається до DWN вузла, який у свою чергу відсилає

Цей запит також надсилається до back-end’у, адже наразі DWN не підтримує складних запитів, що на даний момент робить його дуже не зручним для використання. Після створення посту, користувачі знову потрапляємо на домашню сторінку, де новостворена публікація буде угорі.

Лістинг 3.1 DWN Message для публікацій:

```
interface Post {
  identifier: string //uuid v4 identifier of this content
  author: Person // embedded "light" person schema of the author
  text: string // the post content
  dateCreated: DateTime // date time object for created
  //either generalize like this, or make a part of the CommunityPost/UserPost schema object
  interactions: UserInteraction[] // an array of interactions to this post
}
```

3.4 Сторінка профілю

Коли користувач хоче змінити свій нік, чи фотографію, він переходить на сторінку налаштувань, де він може вказати усі ці дані та обновити профіль у нашій системі, цей виклик буде напряму спрямовано до back-end’у, але у наступних версія застосунку він буде викликати DWN методи, щоб оновлювати глобальний профіль користувача, в усіх сервісах, які використовую децентралізовані вебвузли.

Розглянемо функціонал для редагування налаштувань профілю, який показано на рисунку 3.5.

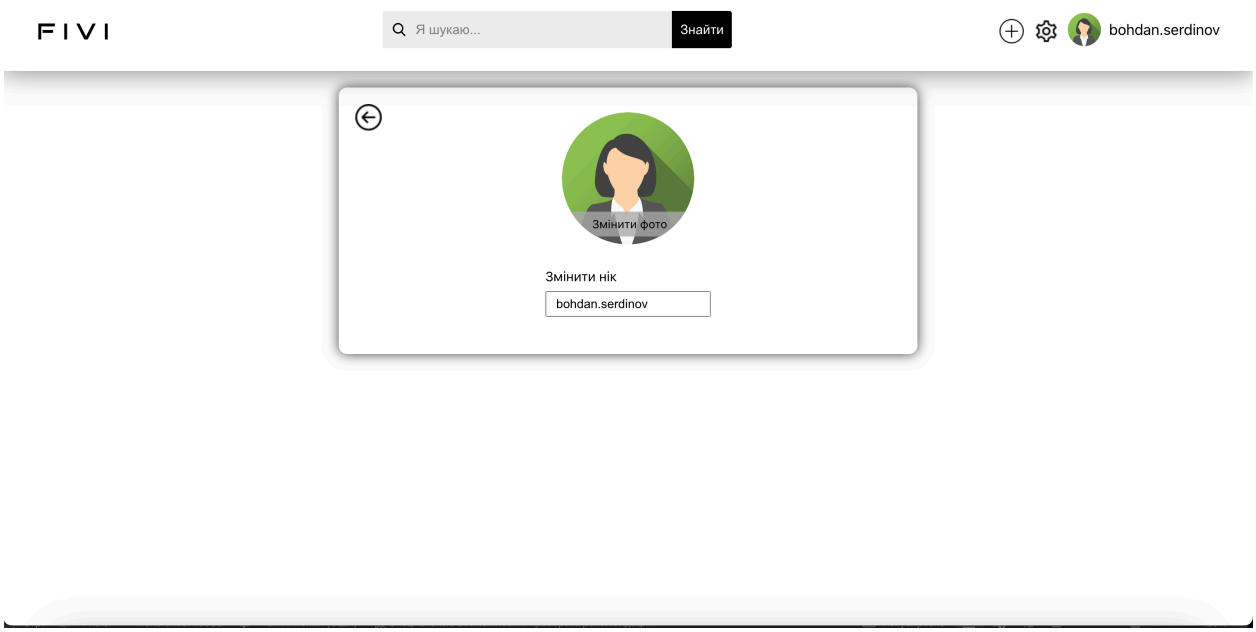

Рисунок 3.5 – Модальне вікно редагування профілю

Так як у кожного користувача є свій профіль, він може переглянути свої публікації коли напискаю на свій нік на домашньому єкрані. Після натискання він побачить такий скрін, який показано на рисунку 3.6.

Для створення сторінки профайлу за ідею було взято соціальну мережу Instagram, яка надає зручний інтерфейс та широкий функціонал для редагування профілю та публікацій.

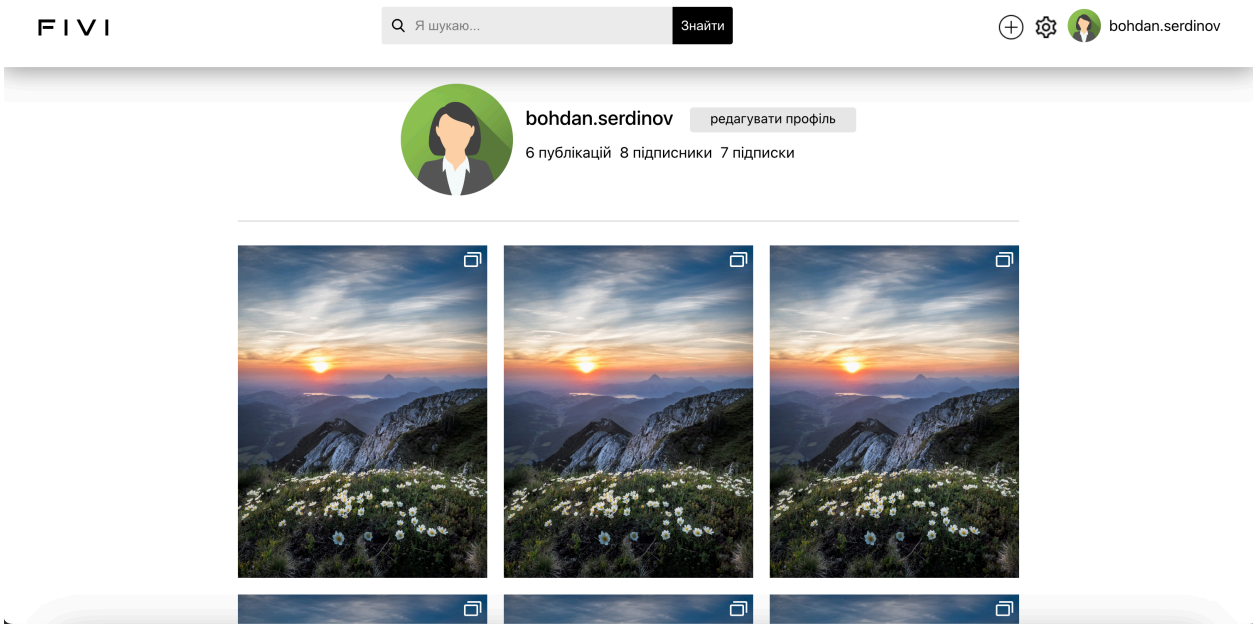

Рисунок 3.6 – Профайл користувача

Також кожну публікацію інші користувачі можуть коментувати, та лайкати. Сторінка публікації показана на рисунку 3.7.

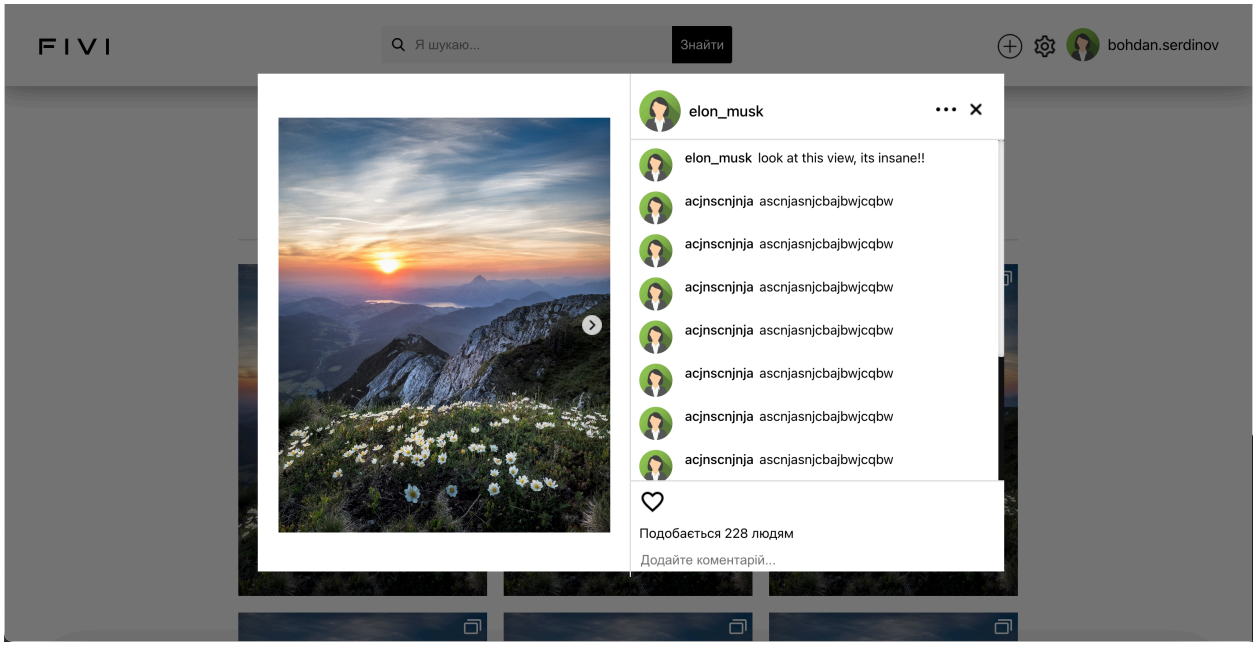

Рисунок 3.7 – Сторінка публікації

Розглянемо функціональні можливості для роботи з коментарями, усі коментарі працюють так саме як й публікації, тобто через DWN запити.

Лістинг 3.2 Реалізація DWN Message для коментаря:

```
interface Comment {
  identifier: string //uuid v4 identifier of this content
  author: Person
  text: string // the comment content
  subjectOf: Post // the post this comment belongs to
  dateCreated: DateTime // date time object for created
  dateModified: DateTime // date time object for edited initial same as created
}
```

Після того як буде сформовано цей месседж, він буде відправлений до DWN вузла, а той у свою чергу надішле цей комментар до back-end'у.

Для того щоб поставити вподобайку на публікацію, потрібно просто натиснути на зображення сердечка. Після того як публікація буде лайкнута, то іконка змінить колір та число лайків збільшиться. Даний процес показаний на рисунку 3.8.

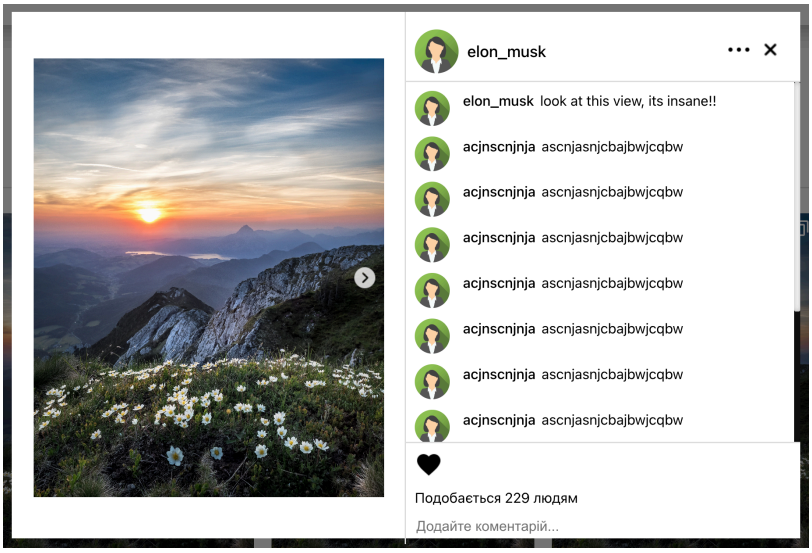

Рисунок 3.8 – Сторінка публікації після того, як користувач поставить лайк

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено застосунок та спроєктовано MVP сервісу децентралізованої соціальної мережі на основі Web3 та Web5 технологій. Розроблений застосунок призначений для користувачів сучасних, централізованих соціальних мереж, які переймаються за розповсюдження своїх даних, недоторканість приватного життя та правові аспекти своєї інформації.

Був проведений аналіз існуючих соціальних мереж, разом з їхньою політикою щодо даних користувачів, що дозволило виділити проблеми з конфіденційністю. Вже існуючі платформи збирають більшість даних юзерів, а згодом на основі цієї вибірки просувають політичну рекламу, маніпулюють бажаннями та потребами, задля більшої фінансової вигоди, що є недопустимим у сучасному житті, адже вже було викрито багато скандальних правопорушень коли соціальні мережі продавали дані користувачів, і на основі цих даних формувалася виборча компанія президентів. Після аналізу ринку децентралізованих соціальних мереж було знайдено досить велику кількість соцмереж, які мають великі проблеми з функціональністю та дизайном. Розроблена система була націлена на вирішення цих проблем, які мають централізовані та децентралізовані системи, на покращення дизайну та розширення функціоналу у децентралізованих соцмережах, та створення логіки для конфіденційного зберігання та обробки інформації у централізованих мережах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Andreas M. Antonopoulos (2017) Mastering Bitcoin. First Edition.

2. Andreas M. Antonopoulos (2017) Mastering Bitcoin: Programming the Open Blockchain. Second Edition.

3. Antony Lewis (2018) The Basics of Bitcoins and Blockchains: An Introduction to Cryptocurrencies and the Technology That Powers Them (Cryptography, Derivatives Investments, Futures Trading, Digital Assets, NFT).

4. Jeff Berwick (2014) The Book Of Satoshi: The Collected Writings of Bitcoin Creator Satoshi Nakamoto.

5. Lorne Lantz, Daniel Cawrey (2020) Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications First Edition.

6. Andreas M. Antonopoulos, René Pickhardt (2022) Mastering the Lightning Network. 1st Ed.

7. Siraj Raval (2016) Книга Decentralized Applications: Harnessing Bitcoin’s Blockchain Technology. First Edition.

8. David Shrier (2020) Basic Blockchain: What It Is and How It Will Transform the Way We Work and Live.

9. Guy Kawasaki, Peg Fitzpatrick (2014) The Art of Social Media.

10. David Kirkpatrick (2014) The Facebook Effect.

11. John Daniel Leon (2018) Security with Go: Explore the power of Golang to secure host, web, and cloud services.

12. V.N. Nikhil Anurag (2018) Distributed Computing with Go.

13. Sachchidanand Singh, Prithvipal Singh (2021) Hands-on Go Programming: Learn Google’s Golang Programming, Data Structures, Error Handling and Concurrency.

14. Matthew A. Titmus (2021) Cloud Native Go: Building Reliable Services in Unreliable Environments. First Edition.

15. Decentralized Web Node specification. URL: https://identity.foundation/decentralized-web-node/spec/ (дата звернення 08.05.2023).

16. Decentralized Identifiers and Decentralized Web Nodes. URL: https://shardeum.org/blog/decentralized-identifiers/ (дата звернення 13.04.2023).

17. Decentralized Web Nodes in Zion Application. URL: https://docs.zion.fyi/Architecture/decentralized-web-nodes (дата звернення 12.04.2023).

18. Bitcoin Lightning Network in Zion Application. URL: https://docs.zion.fyi/Architecture/bitcoin-lightning-network (дата звернення 15.04.2023).

19. Dušan Stojanović (2023) Modern Web Development with Go: Build real-world, fast, efficient and scalable web server apps using Go programming language.

20. Джин Кім, Джез Хамбл, Патрік Дебуа, Джон Вілліс (2023) DevOps. Посібник. Як домогтися гнучкості, надійності і безпеки світового рівня в технічних компаніях.

21. Sebastien Goasguen (2015) Docker Cookbook: Solutions and Examples for Building Distributed Applications 1st Edition.

22. Docker Guides and Docs. URL: https://docs.docker.com/get-started/overview/ (дата звертання 13.05.2023).

23. Dockerizing your Go application. URL: https://blog.logrocket.com/dockerizing-go-application/ (дата звертання 13.05.2023).

24. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Tools for fast metric data search in structural methods for image classification, *IEEE Access*, 10, pp. 124738-124746.

25. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, *Advances in Electrical and Electronic Engineering*, 21(1), pp. 19-27.

26. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С. 5-12.

27. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023) Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, *Сучасні інформаційні системи*, 7(1), С. 5-13.

28. Regina O. Obe (2015) PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition.

29. Neal Ford, Mark Richards, Pramod Sadalage (2021) Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures 1st Edition.

30. Protobuf Guides and Docs. URL: https://protobuf.dev/ (дата звертання 10.05.2023).