МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Розробка краудфандингової платформи»

Виконав студент групи ІПЗ-19-1
ЄВДОКИМОВ Владислав Вікторович

Керівник роботи:
МАРЧУК Галина Вікторівна

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЄВДОКИМОВ Владислав Вікторович**
Керівник роботи: **МАРЧУК Галина Вікторівна**
Тема роботи: **«Розробка краудфандингової платформи»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2023 р., №70с**
Вихідні дані для роботи: **процес краудфандингу в контексті Web 3.0, який використовує технології блокчейн та смарт-контрактів для залучення фінансування та реалізації проектів.**
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1	Галина МАРЧУК	02.03.2023	02.03.2023
2	Галина МАРЧУК	04.04.2023	04.04.2023
3	Галина МАРЧУК	02.05.2023	02.05.2023

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку краудфандингової платформи та пояснювальної записки. Пояснювальна записка до випускної роботи містить 78 сторінок, 53 ілюстрацій та 11 таблиць.

Метою роботи є створення платформи краудфандингу з використанням сучасних технологій та інтеграцією блокчейна для обробки транзакцій та зберіганням даних.

Архітектура платформи краудфандингу базується на блокчейні, що забезпечує прозорість та непорушність зібраних коштів. Смарт-контракти застосовуються для автоматизації процесів збору та розподілу фінансів, створюючи довіру між учасниками платформи.

Додаток має зручний інтерфейс, який дозволяє користувачам легко організовувати збори, встановлювати цілі фінансування та описувати проект.

КЛЮЧОВІ СЛОВА: ВЕБСЕРВІС, БЛОКЧЕЙН, WEB3, ПЛАТФОРМА, СМАРТ-КОНТРАКТ, КРАУДФАНДИНГ, КРИПТОВАЛЮТА.

ABSTRACT

The bachelor's thesis consists of a crowdfunding platform web application and an explanatory note. The explanatory note to the final thesis contains 78 pages, 53 illustrations and 11 tables.

The goal of the work is to create a crowdfunding platform using modern technologies and blockchain integration for transaction processing and data storage.

The architecture of the crowdfunding platform is based on the blockchain, which ensures transparency and inviolability of the collected funds. Smart contracts are used to automate the processes of collecting and distributing finances, creating trust between platform participants.

The app has a user-friendly interface that allows users to easily organize meetings, set funding goals and describe the project.

KEYWORDS: WEBSERVICE, BLOCKCHAIN, WEB3, PLATFORM, SMART CONTRACT, CROWDFUNDING, CRYPTOCURRENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ КРАУДФАНДИНГОВОЇ ПЛАТФОРМИ 7

1.1. Постановка задачі 7

1.2. Аналіз аналогів програмного продукту 9

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 14

Висновки до першого розділу. 19

РОЗДІЛ 2. ПРОЕКТУВАННЯ КРАУДФАНДИНГОВОЇ ПЛАТФОРМИ 20

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи 20

2.2. Розробка бази даних системи 26

2.3. Проектування та реалізація алгоритмів роботи системи 28

2.4. Реалізація функціоналу системи краудфандингової платформи 30

Висновки до другого розділу 38

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КРАУДФАНДИНГОВОЮ ПЛАТФОРМОЮ 40

3.1 Порядок встановлення та налаштування параметрів платформи . 40

3.2 Структура інтерфейсу краудфандингової платформи 41

3.3 Статистика зборів та загальна статистика платформи 51

3.4 Тестування платформи 52

Висновки до третього розділу 57

ВИСНОВКИ 58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59

ДОДАТКИ 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

SDK – набір для розробки програмного забезпечення (software development kit)

MVC – схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete)

DApps – Децентралізований додаток

ВСТУП

Актуальність теми полягає у тому, у використанні протоколу Web 3.0 та сучасних технології блокчейн та смарт-контрактів для створення децентралізованих додатків та платформ. Краудфандинг, в якості однієї з додаткових функціональностей Web 3.0, стає більш прозорим, доступним та безпечним завдяки використанню блокчейн-технологій. Криптовалютний краудфандинг працює аналогічно до традиційного краудфандингу. Краудфандинг криптовалюти дозволяє підприємцям та організаціям підключатися до ширшого кола донорів як на місцевому, так і на міжнародному рівні, які воліють робити пожертвування через мережі Blockchain на підтримку організації та цілей.

Метою випускної кваліфікаційної роботи бакалавра є розробка платформи краудфандингу з використанням технології Web 3.0. Для реалізації поставленої мети потрібно провести огляд функціональності краудфандингових платформ, методів залучення фінансування та статистичних даних, які можуть бути збережені на блокчейні.

Об’єктом дослідження є процес краудфандингу в контексті Web 3.0, який використовує технології блокчейн та смарт-контрактів для залучення фінансування та реалізації проектів.

Предметом дослідження є функціональність децентралізованих платформ краудфандингу на базі Web 3.0, їх можливості та переваги у порівнянні з традиційними централізованими платформами. Дослідження також охоплює аналіз ролі блокчейну у забезпеченні прозорості та безпеки процесу краудфандингу, можливості перевірки використання зібраних коштів та забезпечення довіри між учасниками.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези: Євдокимов В. В., Марчук Г. В. Краудфандингу WEB 3.0. Тези доповідей V Всеукраїнської науково-технічної конференції Комп’ютерні технології: інновації, проблеми, рішення, 01-02 грудня 2022 року. Житомир: «Житомирська політехніка», 2022. С.164-165.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ КРАУДФАНДИНГОВОЇ ПЛАТФОРМИ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення краудфандингової платформи, використовуючи сучасні технології, що дозволить легко користуватись та масштабувати її у майбутньому.

Краудфандинг – це практика фінансування проектів чи підприємства шляхом збору грошей від великої кількості людей, як правило, через Інтернет. Іншими словами краудфандинг це форма краудсорсингу та альтернативне фінансування. Одним з самих відомих випадків краудфандингу є історія Зорана Маджира. Це хорватський винахідник і підприємець, відомим своїм проектом "Pebble Time" смарт-годинником з кольоровим екраном. Він використав краудфандингову платформу Kickstarter для залучення фінансування на реалізацію свого проекту.

У 2012 році Маджир запустив свій перший проект "Pebble", на Kickstarter. Це був смарт-годинник з е-інк-дисплеєм, який мав можливості підключення до смартфона та відображення повідомлень. Проект став надзвичайно популярним і вразив світове співтовариство. Зоран Маджир був спритним у просуванні свого проекту, він створив цікаві відео та розповідав про нього в соціальних мережах. За кілька днів він зібрав понад 10 мільйонів доларів, що значно перевищило його початкову мету в 100 000 доларів.

Краудфандингова платформа має представляти собою декілька програмних модулів, що забезпечують різноманітні функції та можливості для проектів та учасників:

– Модуль збору коштів. Модуль дозволяє проектам створювати сторінки з описом своїх ідей та цілей, встановлювати фінансову мету і тривалість кампанії. Він надає можливість підтримувачам проекту вносити фінансові внески через безпечні платіжні системи.

– Модуль спілкування. Модуль надає засоби для взаємодії між творцями проектів та підтримувачами. Він дозволяє спілкуватися, задавати питання, отримувати оновлення та надсилати подяки підтримувачам.

– Модуль безпеки та гарантій. Модуль забезпечує безпеку фінансових транзакцій, захист від шахрайства та гарантує, що кошти будуть використані згідно зі зобов'язаннями, вказаними творцем проекту.

– Модуль адміністрування. Модуль забезпечує адміністрування додатку, що дає можливість перевіряти кожен збір, переглядати статистику та додавати новий функціонал для платформи.

В результаті повинно бути створено вебсайт на якому будь-який бажаючий може створити збір. Також важливою складовою є децентралізованість, оскільки всі транзакції будуть проводитись використовуючи технологію блокчейн.

Основними етапами розробки платформи є:

1. Розробка дизайну вебсайту:
– створення загальної концепції дизайну додатку;
– створення UI компонентів;
– розробка дизайну сторінок.
2. Побудова архітектури клієнтської частини додатку.
3. Побудова архітектури серверної частини додатку.
4. Розробка системи авторизації та ролей користувачів.
5. Розробка основного функціоналу:
– створення логіки авторизації користувачів з відповідними ролями;
– створення віддаленої бази даних для зберігання інформації;
– імплементація бекенду на клієнтській стороні;
– срганізація спілкування серверної частини з клієнтською;
– створення та інтеграція смарт-контракту.
6. Тестування додатку:
– функціональне тестування додатку;
– тестування графічного інтерфейсу.

Результатом виконання плану дій є створення платформи, за допомогою якої користувачі можуть створювати відповідні збори, які після підтвердження адміністраторами будуть відображатись в загальному списку, що дозволить будь-якому користувачу за допомогою сучасної технології Web 3.0 здійснити переказ коштів за допомогою крипто валюти.

1.2. Аналіз аналогів програмного продукту

Розробка краудфандінгової платформи потребує глибокого аналізу конкурентів на ринку. Оскільки Web 3.0 технологія знаходиться в активній стадії розробки та тестування, проведемо аналіз наявних популярних краудфандингових платформ які працюють на технології Web 2.0.

Одним з таких додатків є платформа Gofundme (рис. 1.1).

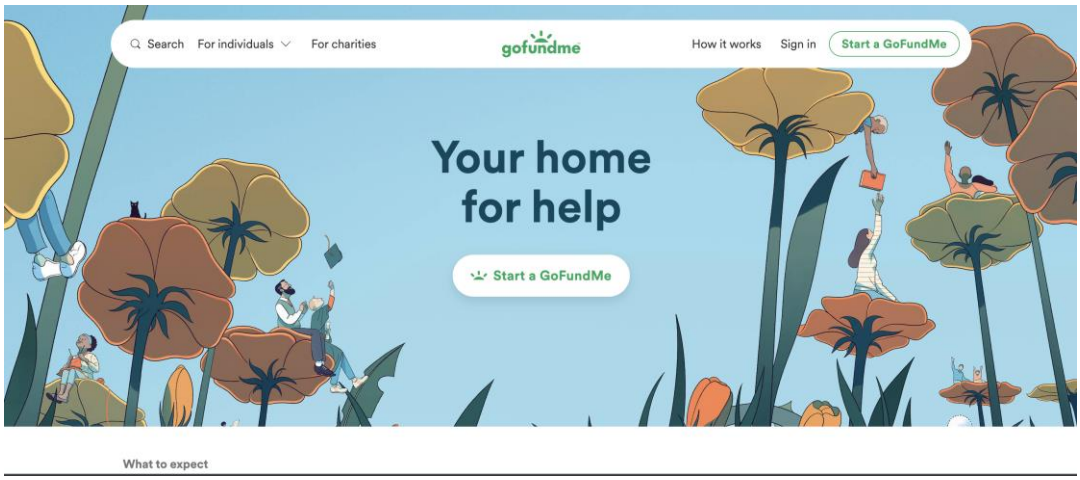

Рис. 1.1. Платформа краудфандингу Gofundme[1]

Дана платформа є одною з провідних на ринку краудфандингових платформ, має гарний дизайн, який є приємним користувачу, будь-який бажаючий може зареєструватися та створити збір.

Основними перевагами Gofundme є:

– інтуїтивно зрозуміла система у користуванні;
– прозорі тарифи;
– має мобільні додатки для IOS і для Android;
– вебдодаткок підтримує різні мови;

– кожен збір має категорію, що дозволяє легко орієнтуватися між зборами;

– кошти можна отримати без збору повної суми.

Недоліками є:

– потребує дуже багато приватної інформації при реєстрації;

– оплата здійснюється тільки банківськими картами.

Kickstarter – популярна платформа для збору коштів. Kickstarter більше орієнтований на творчі проекти, такі як новий музикальний альбом або написання книги, а також на винаходи. Працює за схемою «все або нічого», тобто ви отримуєте гроші тільки в тому випадку, якщо збереться вся необхідна сума (рис.1.2).

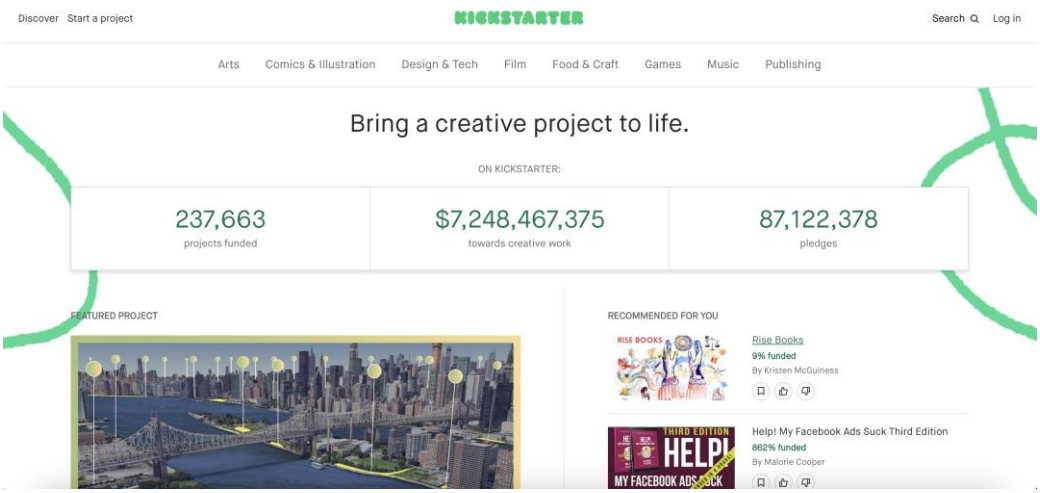

Рис. 1.2. Платформа краудфандингу kickstarter[2]

Основними перевагами kickstarter є:

– вебдодаткок підтримує різні мови;

– проста авторизація.

Недоліками є:

– кошти можна отримати після збору повної суми;

– орієнтована під конкретні категорії;

– має заплутаний дизайн;

– непрозорість тарифів;

– сайт погано оптимізований.

Indiegogo - це платформа масового фінансування (краудфандингу), яка надає можливість людям збирати кошти для реалізації своїх творчих проектів, підприємницьких ідей, соціальних ініціатив та багатьох інших інноваційних проектів. Вона була заснована у 2008 році Деніелом Хедлом і Славко Каваллі в Сан-Франциско, США.

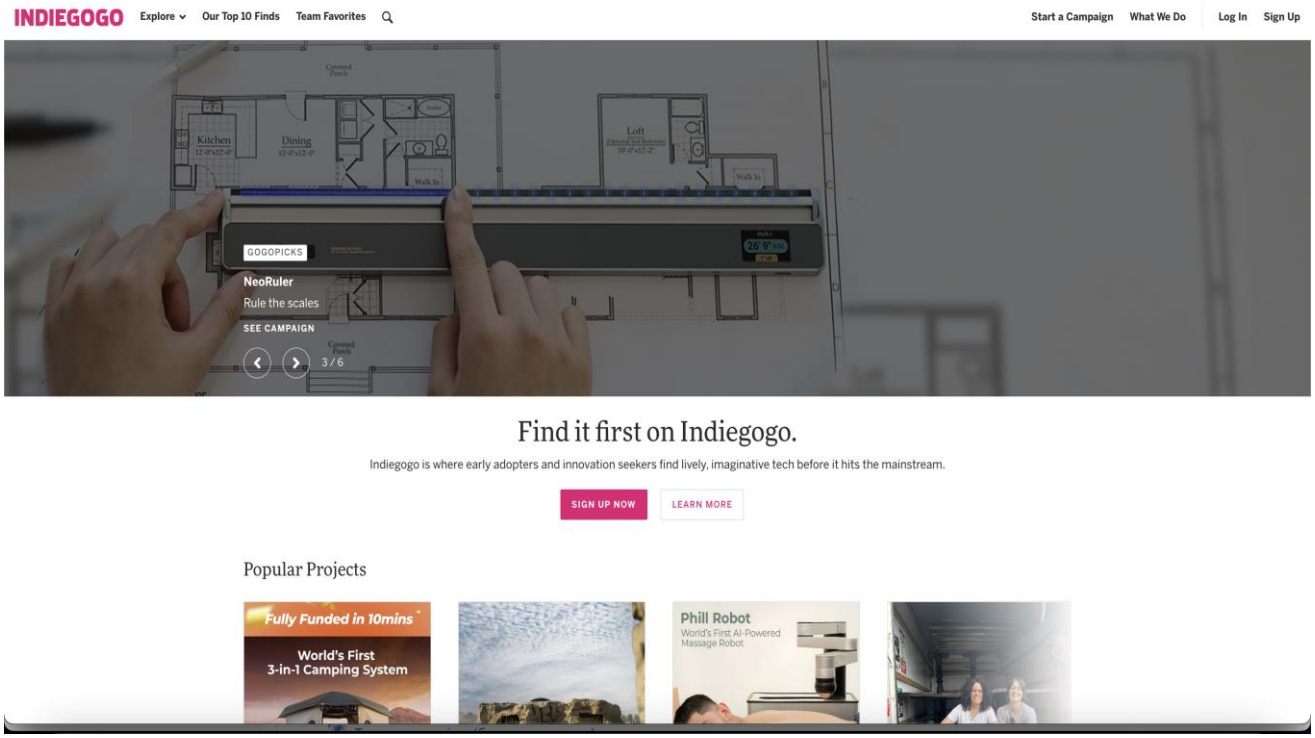

Рис. 1.3. Платформа краудфандингу Indiegogo [3]

Основні переваги Indiegogo:

– глобальна платформа;
– різноманітність проектів;
– гнучкість вибору моделі фінансування;
– можливість взаємодіяти зі спонсорами;
– проста авторизація.

Основні недоліки Indiegogo

– висока конкуренція;
– ризик неуспіху;
– плата за обробку транзакцій;
– правові обмеження;
– застарілий дизайн.

Crowdfunder – є однією з провідних платформ краудфандингу в Великобританії. Основною метою Crowdfunder є допомога людям та організаціям залучити фінансування для реалізації своїх проектів, ідеї чи бізнесу шляхом масового залучення грошової підтримки від спонсорів.

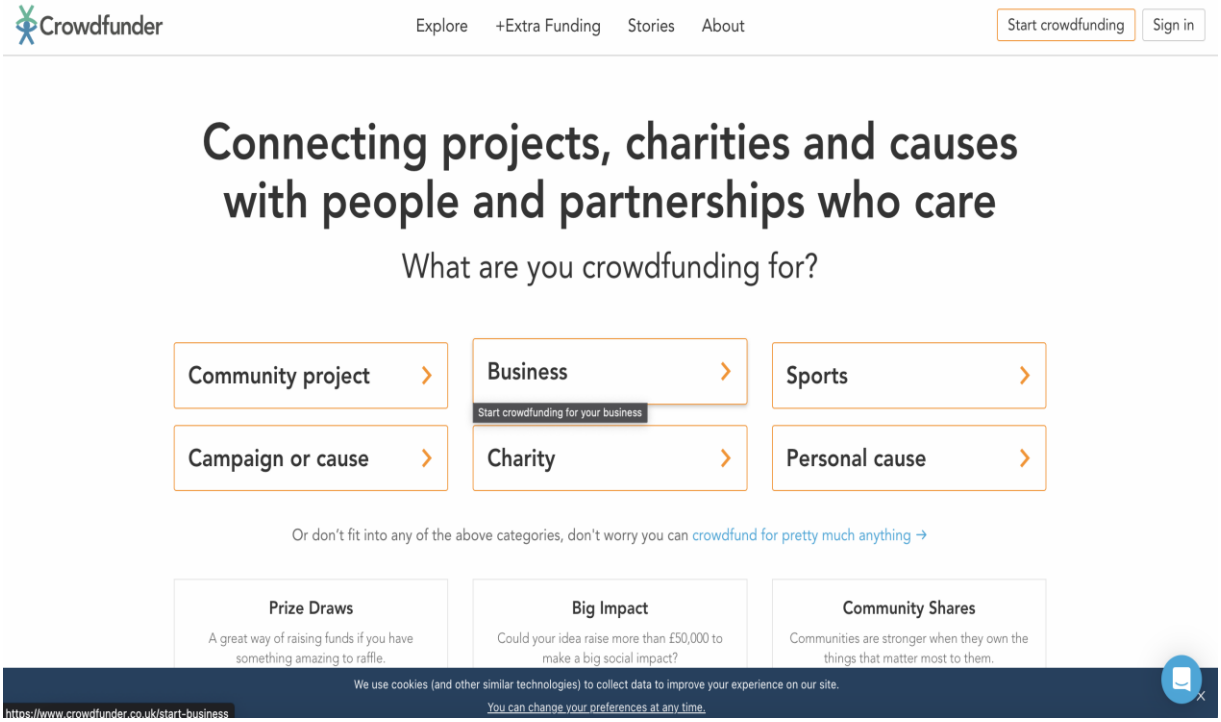

Рис. 1.4. Платформа краудфандингу Crowdfunder[4]

Основні переваги Crowdfunder:

– фокус на Великобританії;
– різні типи проектів;
– нагороди та компенсації;
– ком'юніті-орієнтованість.

Основні недоліки Crowdfunder:

– конкуренція;
– вимоги до проектів;
– втрата часу та ресурсів;
– комісійні витрати;
– застарілий дизайн;
– складна авторизація.

Patreon - на відміну від інших платформ, Patreon є платформою для щомісячної підписки, де прихильники та донори надають регулярні щомісячні внески, мета полягає в тому, щоб забезпечити постійну підтримку творчому підприємству або виконавцю.

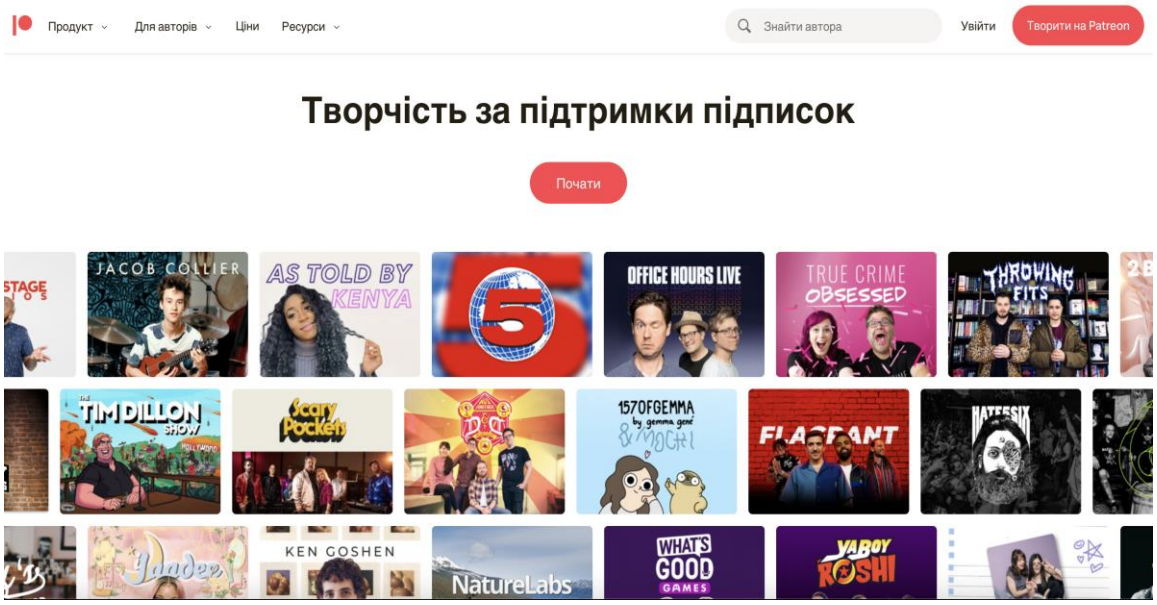

Рис. 1.5. Платформа краудфандингу Patreon[5]

Основними перевагами Patreon є:

– інтуїтивно зрозуміла система у користуванні;

– вебдодаткок підтримує різні мови в тому числі і українську;

– можна оформити підписку на автора і підтримувати його на постійній основі;

– проста авторизація;

Недоліками є:

– оплата здійснюється тільки банківськими картами;

– непрозорість тарифів.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Таблиця порівняння краудфандингових платформ

Характеристика	Gofundme	Kickstarter	Patreon	Indiegogo	Crowdfunder
Прозорість ціни	+	-	+	+	+
Крсоплатформенність	+	-	-	-	-

Продовження таблиці 1.1

Доступність створення збору	+	+	+	+	-
Підтримка будь-якого збору	+	-	+-	+-	+-
Можливість отримати кошти без повного закриття збору	+	-	+	-	-
Зручність використання	+	-	+	+-	+-
Децентралізація	-	-	-	-	-

Переглянувши краудфандингові платформи можна виділити основні відмінності: деякі з них дозволяють отримувати кошти без збору повної суми, деякі дозволяють оформляти підписку, інші дозволяють отримати кошти лише після збору всієї суми, також відрізняються підтримкою зборів, деякі орієнтовані на конкретну аудиторію.

Враховуючи інформацію отриману при дослідженні було прийнято рішення створити децентралізовану платформу, яка не буде залежати від банкової системі світу, та буде побудована на сучасних технологіях з використанням технології Web 3.0, та буде підтримувати будь-які категорії зборів, що дозволить користувачам створювати збір на те, що вони потребують.

Таким чином, даний додаток має бути дуже простим у використанні, з інтуїтивним дизайном; бути децентралізованим, тобто усі платежі будуть здійснюватися за допомогою криптовалюти та технології блокчейн; буде підтримувати будь-які категорії зборів, що дозволить задовольнити вимоги користувачів; отримані кошти будуть зразу потрапляти на рахунок клієнту.

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вебдодаткок буде побудований на клієнт-серверній архітектурі. Серверна частина буде представлена у вигляді API та буде написана з використанням

сучасного бекенд фреймворку NestJS. Для клієнтської частини буде використано фреймворк NextJS, який побудований на основі бібліотеки ReactJS, також для створення смарт-контракту буде використана мова програмування Solidity. Як сховище буде використовуватись база даних Postgress.

NestJS - це фреймворк для розробки серверних застосунків на Node.js з використанням TypeScript. Він базується на концепції архітектурного шаблону Model-View-Controller (MVC), та пропонує багато інших корисних ідей та інструментів для створення високоякісних, масштабованих та легко тестованих додатків [6].

Переваги NestJS:

TypeScript. NestJS написаний на TypeScript, що дозволяє збільшити надійність та зручність розробки за рахунок строго типізованого коду.

Модульність. Фреймворк підтримує модульну архітектуру, що дозволяє легко організовувати код та зменшує його складність.

Dependency Injection. NestJS має вбудований механізм Dependency Injection, що дозволяє легко впроваджувати залежності та забезпечувати високу розширюваність додатків.

Вбудована підтримка WebSocket. Фреймворк надає вбудовану підтримку WebSocket, що дозволяє легко розробляти реально-часні додатки.

Легка тестованість. Завдяки модульній структурі та вбудованим механізмам для тестування, NestJS робить тестування додатків легким та зручним.

Інтеграція з іншими інструментами. NestJS може працювати з багатьма іншими інструментами, такими як Swagger, GraphQL, TypeORM та інші.

Узагальнюючи можна сказати, NestJS є потужним фреймворком для розробки серверних застосунків на Node.js, що дозволяє легко організовувати та підтримувати код, а також забезпечує високу розширюваність та тестованість додатків.

NextJS - це фреймворк для розробки вебдодатків на React. Його особливість полягає в тому, що він надає можливість рендерингу сторінок на

сервері (Server-Side Rendering), а також підтримує статичний генерацію (Static Site Generation) та клієнтський рендеринг (Client-Side Rendering) [7].

Переваги NextJS:

Server-Side Rendering. NextJS дозволяє рендерити сторінки на сервері, що дозволяє збільшити швидкість завантаження сторінок та покращити їх SEO-оптимізацію [8] .

Статична генерація. Фреймворк дозволяє генерувати статичний контент під час збірки додатку, що дозволяє збільшити швидкість завантаження сторінок та зменшити навантаження на сервер.

Клієнтський рендеринг. NextJS підтримує клієнтський рендеринг, що дозволяє створювати динамічний контент та інтерактивні елементи на сторінках.

Вбудована підтримка TypeScript. Фреймворк має вбудовану підтримку TypeScript, що дозволяє збільшити надійність та зручність розробки за рахунок строго типізованого коду.

Легка тестованість. Завдяки вбудованим механізмам для тестування, NextJS робить тестування додатків легким та зручним.

Широкі можливості розширення. Фреймворк підтримує багато різних плагінів та модулів, що дозволяє розширювати його функціональність та забезпечувати більшу гнучкість розробки.

NextJS є потужним фреймворком для розробки вебдодатків на React, що дозволяє використовувати різні методи рендерингу сторінок, забезпечує підтримку TypeScript та легку тестованість. Це робить NextJS дуже привабливим вибором для команд розробників, які шукають швидкий та надійний спосіб розробки вебдодатків на React з високою продуктивністю та зручністю у використанні.

Solidity - це мова програмування для розробки смарт-контрактів (smart contracts) на блокчейні Ethereum. Смарт-контракти - це програми, які автоматично виконують угоди та транзакції на блокчейні без потреби посередника [9, 10, 11].

Основні риси мови Solidity:

Синтаксис. Solidity має синтаксис, подібний до мови програмування C++, що робить її легкою для розуміння і вивчення для розробників з досвідом у C++.

Підтримка контрактів. Solidity підтримує створення різних видів контрактів, включаючи стандартні контракти, бібліотеки, контракти з наслідуванням та інші.

Безпека. Solidity містить декілька вбудованих функцій безпеки, таких як контроль переповнення (overflow control), що допомагають запобігти вразливостям безпеки в смарт-контрактах.

Відкритість. Solidity - це відкрита мова програмування з відкритим вихідним кодом, що дозволяє розробникам співпрацювати та додавати новий функціонал до мови.

Підтримка ERC-20. Solidity підтримує створення токенів на базі стандарту ERC-20, що дозволяє створювати власні токени та проводити ICO.

Solidity дозволяє розробникам створювати смарт-контракти на Ethereum, які можуть взаємодіяти з токенами, гаманцями, децентралізованими біржами та іншими додатками на блокчейні. Завдяки безпеці, простоті та підтримці стандартів, Solidity - це статично типізована мова програмування, розроблена для розробки смарт-контрактів, які працюють на віртуальній машині Ethereum (EVM) або сумісних віртуальних машинах.

Web 3.0 - це нове покоління Інтернету, яке забезпечує зв'язок між користувачами та додатками на блокчейні. Web 3.0 відкриває нові можливості для розробки децентралізованих додатків, що не залежать від централізованих служб, таких як соціальні мережі, пошукові системи та інші.

Web 3.0 передбачає, що користувачі можуть зберігати свої дані, ресурси та інформацію в децентралізованих мережах. Це дає користувачам повний контроль над їхніми даними та дозволяє розробникам створювати додатки, які працюють на базі цих даних.

Web 3.0 також забезпечує можливість проведення безпечних транзакцій, що не потребують посередників та зменшують ризик зловживання. Це може бути

корисно для проведення онлайн-платежів, взаємодії зі смарт-контрактами та проведення інших операцій, що потребують безпеки та довіри.

Web 3.0 також дозволяє розробникам створювати додатки, що можуть взаємодіяти з децентралізованими біржами та іншими фінансовими продуктами на блокчейні. Це відкриває нові можливості для розвитку фінансових технологій та фінансової інклюзії.

У підсумку, Web 3.0 - це нове покоління Інтернету, що відкриває нові можливості для розробки децентралізованих додатків та забезпечує безпеку та довіру в мережі. Що дозволяє користувачам контролювати свої дані та ресурси та взаємодіяти з різноманітними додатками та продуктами на блокчейн.

PostgreSQL, часто називається просто Postgres, - це вільна та відкрита реляційна система управління базами даних (СУБД), що дозволяє зберігати та керувати великими обсягами даних. Є одним з найбільш потужних та розширюваних СУБД на ринку, з високим рівнем надійності та безпеки [12].

Postgres підтримує стандарт SQL та включає в себе багато розширень, що забезпечують розширені функціональні можливості, такі як географічні запити, повнотекстовий пошук, JSON-операції та багато інших. Також підтримує розширення, що дозволяють розробникам створювати власні функції та типи даних.

Однією з найбільш важливих рис Postgres це розширюваність. Також має велику кількість розширень та модулів, що дозволяють розробникам додавати нові функції та можливості. Багато з цих розширень є вільними та відкритими, що дозволяє спільноті розробників постійно вдосконалювати СУБД та додавати нові функції.

Postgres також відомий своєю безпекою та надійністю. Має широкий набір функцій безпеки, таких як шифрування даних, захист від вторгнень та контроль доступу. Також має високий рівень стійкості до помилок та збоїв, що дозволяє забезпечити надійну роботу бази даних.

У підсумку, Postgres - це потужна та надійна СУБД, що має широкий набір функцій та можливостей, яка дозволяє зберігати великі обсяги даних, має

високий рівень безпеки та надійності, підтримує стандарт SQL та має багато розширень та модулів. Postgres є популярним вибором для багатьох видів додатків та проектів, від малих вебсайтів до великих корпоративних систем.

Висновки до першого розділу.

Проведено аналіз предметної області за допомогою якого були визначені основні вимоги до платформи краудфанднгу, основною задачею якої є створення зборів користувачами для відповідних цілей.

В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані та порівняні різні платформи з різними принципами роботи, основним функціоналом яких є створення зборів або підтримка авторів місячною підпискою. Крім цього, були сформовані основні вимоги для власної платформи.

На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки вебдодатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ КРАУДФАНДИНГОВОЇ ПЛАТФОРМИ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Краудфандингова платформа надає вебінтерфейс, через який користувачі можуть створювати кампанії, збирати кошти та взаємодіяти один з одним. Функціональність краудфандингової системи залежить від активності користувачів. Користувачі можуть бути розділені на дві основні групи: творці збору, які шукають фінансування для своїх проектів, та спонсори, які внесуть фінансовий внесок на підтримку проектів. Збори містять деталі про проект, його цілі, фінансові цілі, строк збору коштів та інші важливі відомості, які можуть зацікавити потенційних спонсорів. Для наглядної взаємодії між елементами додатку було створено Діаграму варіантів використання (рис 2.1).

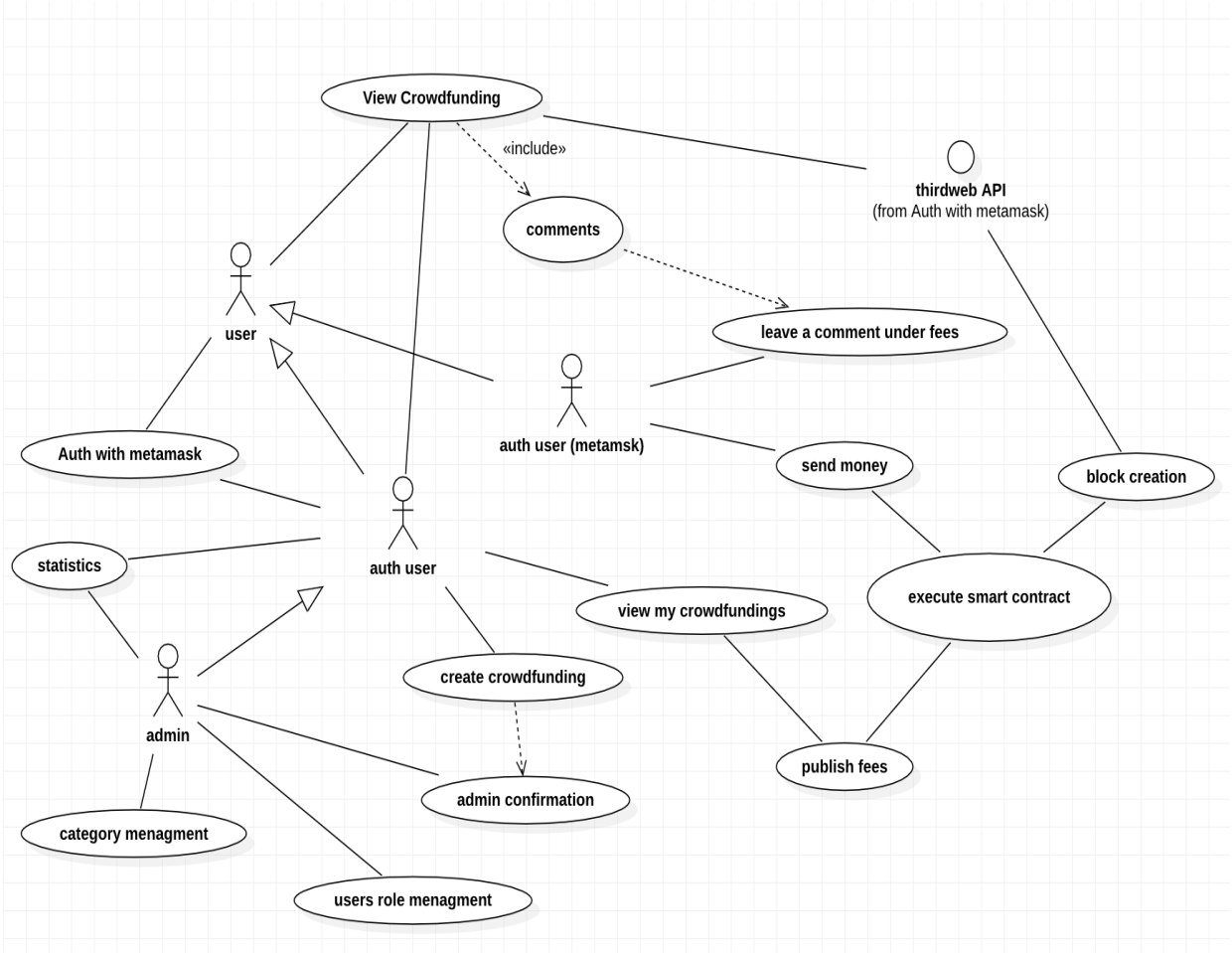

Рис. 2.1. Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.7.

Таблиця 2.1

Опис прецеденту «Вхід/Реєстрація»

Назва варіанту використання	Вхід/реєстрація
Основні актори	Адміністратор, власник кампанії, спонсор
Короткий опис	Користувачі можуть зареєструватися, щоб створити новий збір, також є можливість авторизації через метамаск для спонсорів, та авторизація для адміністраторів, щоб керувати зборами

Таблиця 2.2

Опис прецеденту «Створення збору»

Назва варіанту використання	Створення збору
Основні актори	Адміністратор, власник кампанії, спонсор
Короткий опис	Користувачі можуть створювати свої власні кампанії на платформі для збору фінансування.

Таблиця 2.3

Опис прецеденту «Пошук та перегляд зборів»

Назва варіанту використання	Пошук та перегляд зборів
Основні актори	Спонсор, користувач
Короткий опис	Користувачі можуть шукати та переглядати різні кампанії, які були створені на платформі.

Таблиця 2.4

Опис прецеденту «Внесення фінансового внеску»

Назва варіанту використання	Внесення фінансового внеску
Основні актори	Спонсор
Короткий опис	Спонсори мають можливість здійснювати фінансові внески до кампаній, які вони підтримують.

Таблиця 2.5

Опис прецеденту «Комунікація та взаємодія»

Назва варіанту використання	Комунікація та взаємодія
Основні актори	Спонсор, користувач
Короткий опис	Користувачі можуть взаємодіяти один з одним, коментувати кампанії, задавати питання та обговорювати ідеї.

Таблиця 2.6

Опис прецеденту «Управління кампаніями»

Назва варіанту використання	Управління кампаніями
Основні актори	Власник кампанії
Короткий опис	Творці кампаній можуть керувати своїми проектами, встановлювати цілі фінансування, створювати нові кампанії

Таблиця 2.7

Опис прецеденту «Аналітика та звітність»

Назва варіанту використання	Аналітика та звітність
Основні актори	Власник кампанії, адміністратор
Короткий опис	Платформа надає аналітичні засоби та звіти для відстеження фінансових показників та успішності кампаній.

Основна ціль розробки краудфандингової платформи полягає в створенні онлайн-середовища, яке дозволяє збирати фінансову підтримку від широкої аудиторії для реалізації різноманітних проектів. Платформа спрямована на забезпечення ефективної комунікації та взаємодії між творцями проектів та потенційними спонсорами, а також на надання зручного і надійного механізму для збору фінансування.

Основна мета полягає у сприянні здійсненню ідей, творчих проектів, благодійних ініціатив та підприємницьких починань шляхом залучення коштів з

числа зацікавлених спонсорів. Платформа надає можливість творцям проектів представити свої ідеї, описати цілі, нагороди для спонсорів та показувати прогрес робіт. Завдяки цьому, основною метою є створення умов для успішного збору необхідних коштів та підтримки проектів, які можуть мати значний вплив на суспільство, культуру, науку та бізнес.

Вимоги користувачів:

Внутрішній користувач – адміністратор:

1. Спілкування з авторами зборів.
2. Підтвердження створення зборів.
3. Перегляд заявок на надання послуги.

Внутрішній користувач – клієнт:

1. Авторизація.
2. Перегляд власних зборів.
3. Перегляд статусу сотворених зборів.
4. Створення нової заявки на надання послуг.
5. Перегляд списку зборів.
6. Перегляд статистики.

Внутрішній користувач – гість:

1. Перегляд списку зборів.
2. Анонімна авторизація.

Внутрішній користувач – гість (metamask):

1. Перегляд списку зборів.
2. Анонімна авторизація.
3. Створення коментарів під збором.
4. Перерахування коштів на збір.

Функціональні вимоги:

1. В системі повинна бути представлена можливість авторизації користувачів, двома способами, за допомогою технологій Web 3.0 та звичайна авторизація.
2. Забезпечення анонімності звичайним користувачам.

3. Забезпечення спілкування користувачів.

4. Відображення результатів зборів.

5. Можливість збереження інформації про збори.

6. Забезпечити власникам зборів переглядати статистику.

Не функціональні вимоги:

1. Сприйняття:

– час, потрібний для освоєння вебсайту – від 3-х до 5-ти хвилин, для досвідчених менше 2-х хвилини;

– час відповіді вебсайту на дії користувача не повинні перевищувати 3 секунд, а для складних виконання смарт-контрактів не більше – 90-ти секунд;

– інтерфейс вебсайту повинен бути інтуїтивно зрозумілим та зручним для користувача, та не вимагати від нього додаткової підготовки.

2. Надійність:

– вебсайт має бути доступний на протязі усієї доби.

3. Продуктивність:

– велика кількість користувачів мають змогу переглядати вебресурс одночасно.

Системні вимоги:

1. Вимоги до середовища виконання: платформа повинна задовільнять вимогам на пристроях, що знаходиться в наступній мінімальній комплектації:

– 2 Гб оперативної пам’яті.

– Процесор з тактовою частотою не нижче 1.4 GHz.

– Вихід в мережу інтернет.

2. Вимоги до серверу інформаційної системи:

– У ядрі системи повинна бути документаційно орієнтована база даних у зв’язці з Web 3.0 орієнтованим вебдодатком для забезпечення анонімної авторизації користувачів та виконання смарт-контрактів.

– Підтримка сервером технологій NodeJS.

Аналіз функціональних вимог дозволив виділити наступні сутності, що забезпечать програмну реалізацію додатку. На рисунку 2.2. наведено Діаграму класів, яка використовуються для його роботи.

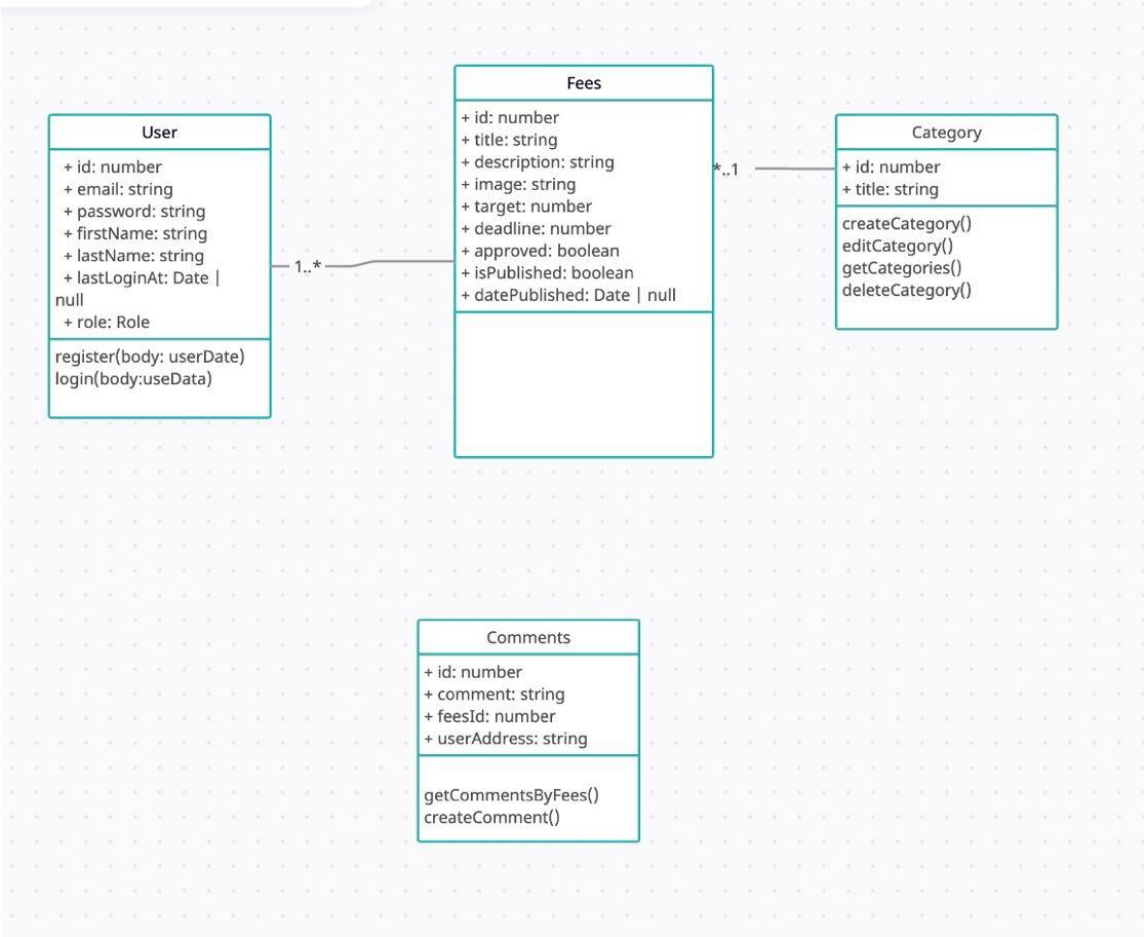

Рис. 2.2. Діаграма класів

User – клас який відповідає за користувача:

– register() – відповідає за реєстрацію;
– login() – відповідає за авторизацію.

Fees – клас який відповідає за збір:

– getFeeses() – відповідає за отримання всіх зборів адміністратором;
– getMyFeeses() – відповідає за отримання всіх зборів користувача;
– createFees() – відповідає за створення збору користувачем;
– aproveFees() – відповідає за підтвердження збору адміністратором;
– publishFees() – відповідає за статус публікування збору.

Category – клас який відповідає за категорії:

– createCategory() – відповідає за створення категорії;

– editCategory() – відповідає за редагування категорії;

– getCategories() – відповідає за отримання категорій;

– deleteCategory() – відповідає за видалення категорії.

Comments – клас який відповідає за коментарі:

– createComment() – відповідає за створення коментаря;

– getCommentsByFees() – відповідає за отримання коментарів відповідного збору.

2.2. Розробка бази даних системи

Для збереження файлів та інформації у додатку використовуються PostgreSQL, це система управління базами даних, яка базується на реляційної моделі. Також після публікації збору, він зберігається на блокчейні, там же зберігаються і всі транзакції.

База даних системи складається з п’яти таблиць:

1. Users – таблиця, яка зберігає данні користувачів;
2. Comments – таблиця, яка зберігає коментарі;
3. Category – таблиця, яка зберігає категорії;
4. Fees – таблиця, яка зберігає збори.

Опис кожної з таблиць бази даних наведений у таблицях 2.8 – 2.11.

Таблиця 2.8

Опис таблиці Users

Поле	Тип	Призначення
id	int	Ідентифікатор
email	varchar	Email користувача
password	varchar	Пароль користувача
firstName	varchar	Ім`я
lastName	varchar	Фамілія
lastLoginAt	date	Дата останнього логіна
Role	enum	Роль

Таблиця 2.9

Опис таблиці Category

Category		
Поле	Тип	Призначення
id	int	Ідентифікатор
title	varchar	Назва категорії

Таблиця 2.10

Опис таблиці Comments

Comments		
Поле	Тип	Призначення
id	int	Ідентифікатор
comment	varchar	Коментар
feesId	int	Айді збору в блокчейні
userAddress	varchar	Адреса гаманця користувача

Таблиця 2.11

Опис таблиці Fees

Fees		
Поле	Тип	Призначення
id	int	Ідентифікатор
title	varchar	Заголовок збору
description	varchar	Опис збору
image	varchar	Посилання на картинку
target	float	сума збору
deadline	int	Кількість днів яку триватиме збір
approved	boolean	Підтвердження адміністрацією
isPublished	boolean	Статус чи опублікований збір
datePublished	date	Дата публікації

У контексті блокчейн-технологій, дані після створення смарт-контракту зберігаються в самому блокчейні. Блокчейн є розподіленою базою даних, яка зберігається на різних вузлах (комп'ютерах) в мережі. Кожен блок у блокчейні містить транзакції, які включають в себе дані, пов’язані зі створенням, оновленням або виконанням смарт-контрактів.

Коли смарт-контракт створюється, дані, які він містить, включаючи його код та параметри, включаються до транзакції. Ця транзакція підписується власником контракту та надсилається до мережі блокчейну для обробки. Коли транзакція підтверджується мережею та включається до блоку, смарт-контракт стає доступним для виконання.

Після створення смарт-контракту та його включення до блокчейну, дані, які він містить, стають публічно доступними і не можуть бути змінені без виконання відповідних транзакцій або методів контракту. Це забезпечує надійність і незмінність даних, оскільки блокчейн є недоступним для зовнішнього втручання.

Для звернення до даних смарт-контракту використовуються спеціальні методи або функції, які надаються самим контрактом. Ці методи дозволяють отримувати, змінювати або видаляти дані, що зберігаються в контракті. Використовуючи ці методи та комунікуючи з контрактом через транзакції, учасники мережі можуть взаємодіяти з даними смарт-контракту та виконувати необхідні операції.

2.3. Проектування та реалізація алгоритмів роботи системи

Додаток є простим у використанні, має всі основні функції платформи, шо дозволяє користувачам легко взаємодіяти з платформою. Алгоритм роботи з платформою зображено на рисунку 2.3, також описано детально його пункти.

Якщо авторизувався адміністратор:

– після авторизації менеджер потрапляє в адміністративну панель;

– в панелі адміністратор може переглянути всі заявки і підтвердити їх, якщо вони задовольняють умови користування платформою;

– адміністратор має можливість створювати нові категорії, редагувати категорії, та видаляти якщо потрібно;

– також адміністратор має можливість додавати нових адміністраторів шляхом змінення ролі.

Якщо авторизувався клієнт:

– клієнт має можливість переглянути список активних зборів;

– клієнт має можливість переглянути статистику по окремим зборам;

– клієнт має можливість створити нову заявку на надання послуг;

– клієнт має можливість опублікувати збори які були підтверджені адміністрацією.

Якщо авторизувався користувач:

– користувач має можливість переглянути активні збори;

– користувач має можливість анонімно переказати певну кількість коштів;

– користувач має можливість залишити коментар під збором.

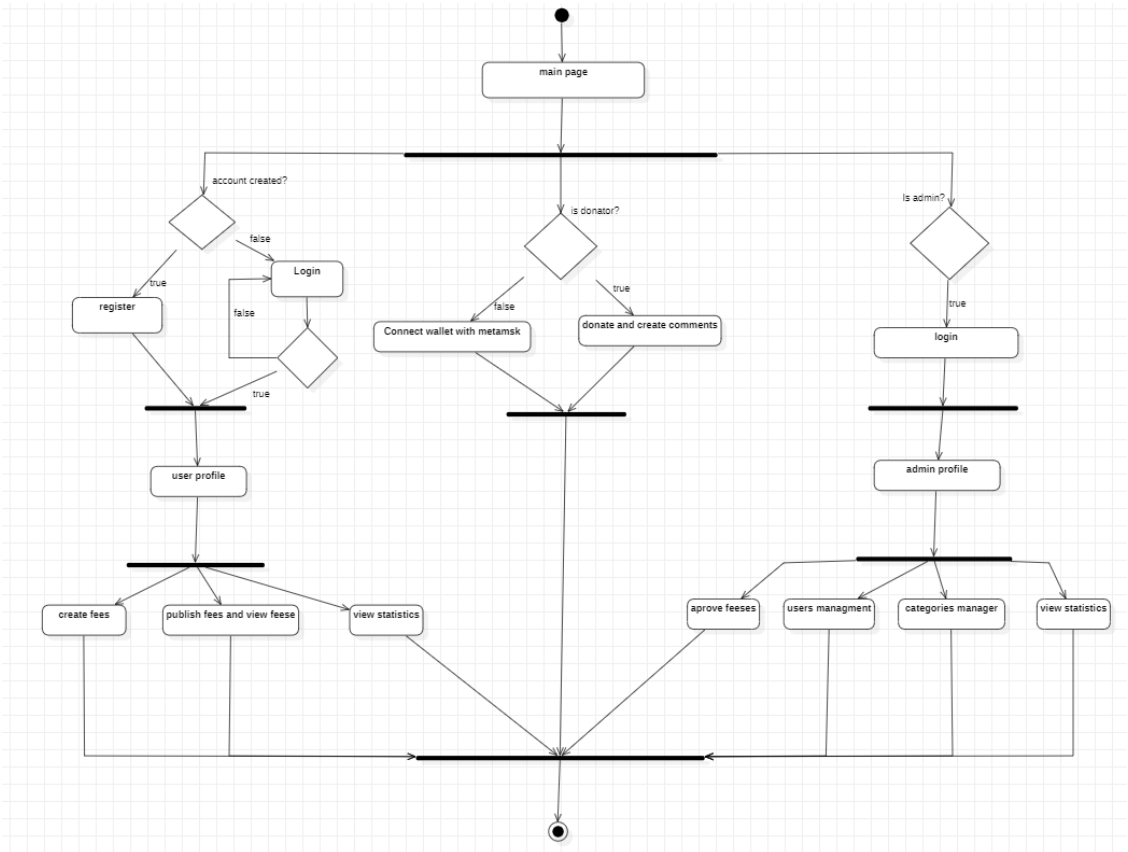

Рис. 2.3. Діаграма активності

2.4. Реалізація функціоналу системи краудфандингової платформи

Розглянемо основні етапи створення збору на платформі. Для цього користувач повинен здійснити реєстрацію нового аккаунту після чого він має можливість створити новий збір. Функція створення збору представлена на рисунку 2.4.

```
public async createFees(body: CreateFees, currentUser: User) {
  const fees = new Fees();
  Object.assign(fees, body);
  fees.category = body.category;
  fees.user = currentUser;
  return await this.repository.save(fees);
}
```

Рис. 2.4. Створення нового збору

Після створення збору має відбутись підтвердження його адміністрацією, для цього створена окрема функція (рис. 2.5).

```
public async aproveFees(id: number) {
  const fees = await this.repository.findOne({ where: { id: id } });
  if (!fees) {
    throw new HttpException('Not found', HttpStatus.NOT_FOUND);
  }
  return this.repository.update(id, { approved: true });
}
```

Рис. 2.5. Підтвердження збору адміністрацією

Після чого користувач може переглянути в особистому кабінеті збори, які були підтверджені адміністрацією, і може їх опублікувати, для цього існує два інструмента які відповідають за це. Перший це створення збору за допомогою смарт контракту з яким будуть взаємодіяти всі користувачі. В даному випадку створення збору на бекенді потрібно тільки для його валідації адміністрацією. Після підтвердження все дії відносно збору будуть проводитись через блокчейн, це публікація збору, транзакції, отримання відповідних даних для підрахунку статистики.

Функція, яка відмічає збір як опублікований, потрібна для того, щоб користувач не міг опублікувати збір два рази (рис. 2.6).

```
public async publishedFees(id: number, currentUserId: number) {
  const fees = await this.repository.findOne({ where: { id: id }, relations: { user: true }
  if (!fees) {
    throw new HttpException('Not found', HttpStatus.NOT_FOUND);
  }

  if (fees.user.id !== currentUserId) {
    throw new HttpException('Your not author', HttpStatus.FORBIDDEN);
  }

  return this.repository.update(id, { isPublished: true, datePublished: new Date() });
}
```

Рис. 2.6. Публікація збору

Більше детально розглянемо функціонал смарт-контракту. У смарт-контракті передбачено створення збору, переказ коштів на збір, отримання транзакцій по збору, отримання збору по ідентифікатору, та отримання всіх зборів.

Першим етапом створення смарт-контракту це створення його полів з якими будуть взаємодіяти його функції (рис 2.7).

```
struct Campaign {
    uint256 id;
    address owner;
    string title;
    string description;
    uint256 target;
    uint256 deadline;
    uint256 amountCollected;
    string image;
    uint256 category;
    address[] donators;
    uint256[] donations;
}

mapping(uint256 => Campaign) public campaigns;

uint256 public numberOfCampaigns = 0;
```

Рис. 2.7. Основні поля контракту

Після чого на основі цих полів створюються функції за допомогою яких можна буде взаємодіяти з ними.

Функція створення збору. Функція приймає поля для створення збору, після чого проходить базова перевірка на дату його закінчення, потім створюється екземпляр збору і присвоюється йому унікальний ідентифікатор і данні які передав користувач, щоб в майбутньому можна було отримувати данні по конкретному збору (рис. 2.8).

```
function createCampaign(
    address _owner,
    string memory _title,
    string memory _description,
    uint256 _target,
    uint256 _deadline,
    uint256 _category,
    string memory _image
) public returns (uint256) {
    uint256 newId = numberOfCampaigns;
    Campaign storage campaign = campaigns[newId];

    require(
        _deadline > block.timestamp,
        "The deadline should be a date in the future."
    );
    campaign.id = newId;
    campaign.owner = _owner;
    campaign.title = _title;
    campaign.description = _description;
    campaign.target = _target;
    campaign.deadline = _deadline;
    campaign.amountCollected = 0;
    campaign.image = _image;
    campaign.category = _category;

    numberOfCampaigns++;

    return newId;
}
```

Рис. 2.8. Створення збору

Після публікації збору, користувачі можуть взаємодіяти з ним використовуючи унікальний ідентифікатор збору.

Функція отримання збору або всіх зборів. Ці дві функцію працюються як GET запит на серверній частині, можна отримати всі збори або якийсь конкретний. Для отримання всіх зборів створюється масив і в нього записуються всі збори. Для отримання конкретного збору, виконується пошук по ідентифікатору (рис 2.9).

Ці дві функції потрібні для того щоб була можливість взаємодіяти з блокчейном, оскільки після створення зборів подальші дії виконуються через блокчейн, що забезпечує децентралізованість додатку, та прозорість виконання транзакцій, оскільки код контракту знаходиться в публічному доступі, що дозволяє досвідченим користувачам перевірити прозорість виконання всіх дій.

```
function getCampaigns() public view returns (Campaign[] memory) {
    Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);

    for (uint i = 0; i < numberOfCampaigns; i++) {
        Campaign storage item = campaigns[i];

        allCampaigns[i] = item;
    }

    return allCampaigns;
}

function getCampaignById(
    uint256 _id
) public view returns (Campaign memory) {
    require(_id < numberOfCampaigns, "Invalid campaign ID.");

    return campaigns[_id];
}
```

Рис. 2.9. Отримання зборів

Головна функція - це переказ коштів на збір. Вона теж знаходиться в смарт-контракті, оскільки проходить взаємодія з блокчейном та конкретним збором який там розміщений. Надіслані кошти відразу потрапляють у гаманець власника збору, що дозволяє користувачу, який анонсував збір, в будь-якому випадку отримати кошти, та унеможливлює людський фактор зі сторони платформи, що також робить додаток децентралізованим (рис 2.10).

```
function donateToCampaign(uint256 _id) public payable {
    uint256 amount = msg.value;

    Campaign storage campaign = campaigns[_id];

    require(
        campaign.deadline > block.timestamp,
        "The deadline for this campaign has already passed."
    );

    campaign.donators.push(msg.sender);
    campaign.donations.push(amount);

    (bool sent, ) = payable(campaign.owner).call{value: amount}("");

    require(sent, "Failed to send Ether to campaign owner.");

    campaign.amountCollected += amount;
}
```

Рис. 2.10. Переказ коштів на збір

Розглянемо як відбувається взаємодія зі смарт-контрактом на клієнтській частині. Для взаємодії зі смарт-контрактом використовується бібліотека thirdweb та реалізовані відповідні функції в React Context для зручного використання в будь-якому місці додатку [13-20].

Для того щоб використовувати функції смарт-контракту потрібно використати хук з SDK thirdweb, за допомогою нього встановлюємо з'єднання з контрактом який розгорнутий в мережі і до якого є публічний доступ.

```
const { contract } = useContract(process.env.NEXT_PUBLIC_CONTRACT);
```

Рис. 2.11. Підключення до контракту

Тепер у нас є екземпляр контракту і можна використати його методи для обміну даними з блокчейном.

Для публікації збору була створена функція, яка приймає в себе аргументи у вигляді масиву, це є особливістю смарт-контрактів вони приймають аргументи не об'єктом, а у вигляді масиву (рис.2.12).

```
const publishCampaign = async (form: CreateCampaign) => {
  try {
    const data = await createCampaign({
      args: [
        address,
        form.title,
        form.description,
        form.target,
        form.deadline,
        form.category,
        form.image,
      ],
    });

    return data;
  } catch (error) {
    console.log("contract call failure", error);
  }
};
```

Рис. 2.12. Створення збору

Для отримання даних про збори створена функція, яка за допомогою методу смарт-контракта отримує данні з блокчейну після чого відбувається парсинг даних оскільки блокчейн має певні особливості в їх зберіганні. Всі числа зберігаються у форматі BigNumber, тому для відображення даних зрозумілих користувачу робимо парсинг цих даних. Що стосується даних про оплати вони теж зберігаються у специфічному форматі який має назву WEI [21], тому для парсингу цих даних використовується бібліотека ethers [22] і за допомогою функції formatEther [23] даної бібліотеки приводимо дані до числового формату, який зрозумілий користувачу(рис.2.13).

```
const getCampaigns = async () => {
  const campaigns = await contract?.call("getCampaigns");

  const parsedCampaings = campaigns?.map((campaign: any, i: any) => ({
    pId: campaign.id.toNumber(),
    owner: campaign.owner,      You, last month • add web3.0 context and conn
    title: campaign.title,
    description: campaign.description,
    goal: ethers.utils.formatEther(
      campaign.target.sub(campaign.target.mod(1e14))
    ),
    deadline: campaign.deadline.toNumber(),
    raised: ethers.utils.formatEther(
      campaign.amountCollected.sub(campaign.amountCollected.mod(1e14))
    ),
    image: campaign.image,
    category: campaign.category.toNumber(),
  }));

  return parsedCampaings?.filter(
    (item: any) => item.deadline > new Date().getTime()
  );
};
```

Рис. 2.13. Отримання зборів

Для переказу коштів збору створено функцію, яка в аргументах приймає ідентифікатор збору та суму для переказу, також дана функція приймає адресу гаманця для переказу комісії. Гаманець заданий у вигляді змінної оточення, щоб за потреби можна було змінити гаманець(рис.2.14).

```
const donate = async (pId: any, amount: any) => {
  try {
    const data = await contract?.call(
      "donateToCampaign",
      [pId, process.env.NEXT_PUBLIC_ADDRESS],
      {
        //@ts-ignore
        value: ethers.utils.parseEther(String(parseFloat(amount))),
      }
    );
    return data;
  } catch {
    return;
  }
};
```

Рис. 2.14 Переказ коштів

Для отримання статистики також була створена функція представлена на рисунку 2.15. Для статистики отримуємо великий об’єм даних, які потрібно перетворити оскільки більшість з них зберігається у вигляді масивів та чисел у форматі BigNumber. У відповідь отримаємо інформацію про збір, кількість донаторів, їхні гаманці, кількість донатів, суми переказів, та комісію яку отримала платформа з кожного донату, в циклі ми розподіляємо основні пункти по змінним також отримуємо адресу гаманця автора, а також данні про збір, після чого за допомогою ще одного циклу ми перетворюємо данні донатів та комісії. В кінцевому результаті отримаємо масив об’єктів, де є інформація про збір, про всі перекази та всі комісії які отримала платформа. Це дозволяє отримати актуальну інформацію по отриманим коштам платформою та кількістю донатів. Наприклад, можна визначити загальну суму переказів, середню суму переказу, медіану, максимальну та мінімальну суми переказів, кількість унікальних донаторів, загальну суму комісій тощо.

Цей підхід до збору та аналізу статистики надає детальну інформацію про збір коштів, його обсяг та структуру. Крім того, він дозволяє зробити різні висновки та статистичні аналізи, що допомагає краще розуміти процес збору коштів та його результати.

```
const getAllDonationsForCampaigns = async () => {
  const data = await contract?.call("getAllDonationsForCampaigns");

  const parsedData = [];
  if (data) {
    for (let i = 0; i < data[0].length; i++) {
      const campaign = data[0][i];
      const author = data[1][i];
      const campaignDonators = data[2][i];
      const campaignDonations = data[3][i];
      const campaignCommissions = data[4][i];

      const parsedCampaign: any = {
        id: campaign.id,
        title: campaign.title,
        owner: author,
        goal: campaign.target,
        deadline: campaign.deadline.toNumber(),
        raised: ethers.utils.formatEther(
          campaign.amountCollected.sub(campaign.amountCollected.mod(1e14))
        ),
        category: campaign.category.toNumber(),
        donations: [],
      };

      for (let j = 0; j < campaignDonators.length; j++) {
        const donator = campaignDonators[j];
        const donation = ethers.utils.formatEther(
          campaignDonations[j].toString()
        );
        const commission = ethers.utils.formatEther(
          campaignCommissions[j].toString()      You, last week • statistics …
        );

        parsedCampaign.donations.push({ donator, donation, commission });
      }

      parsedData.push(parsedCampaign);
    }
  }
  return parsedData;
};
```

Рис. 2.15 Отримання статистики

Підсумовуючи, було створено діаграму компонентів краудфандингової платформи (рис. 2.16) і визначено основні компоненти збору.

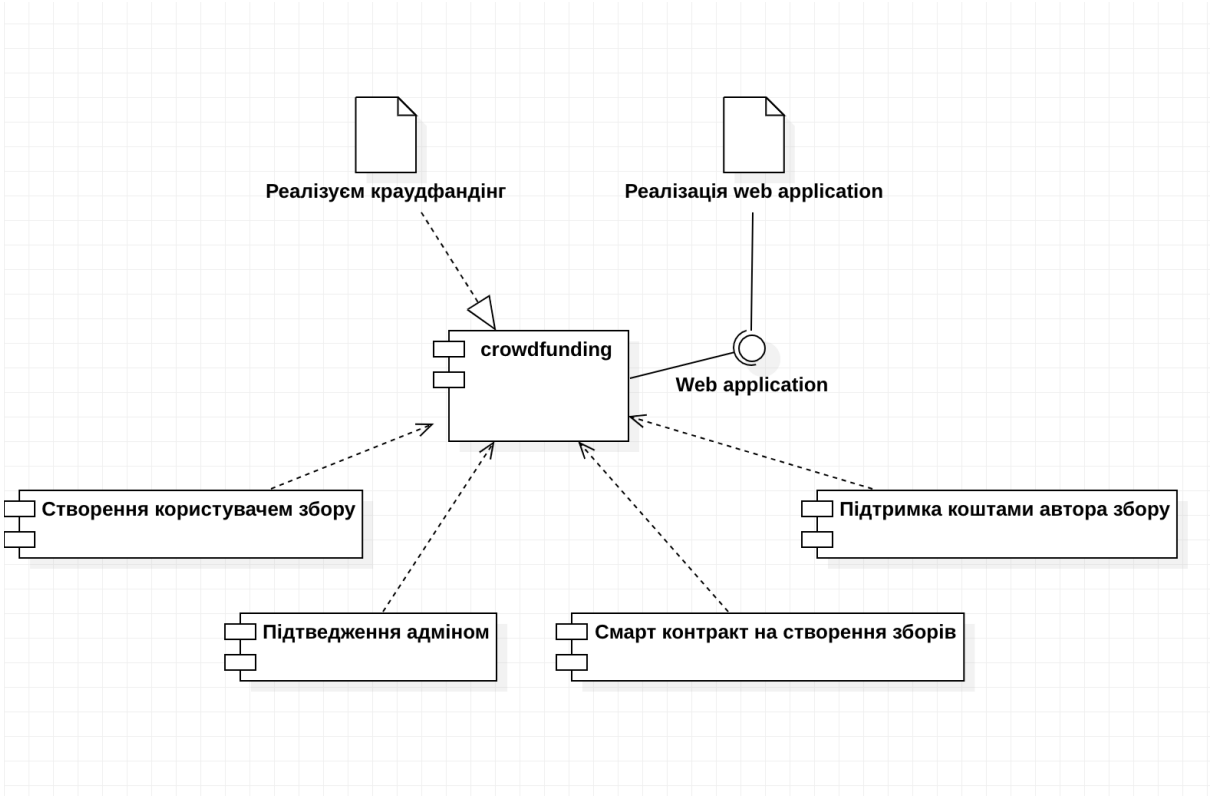

Рис. 2.16. Діаграма компонентів

На діаграмі визначені такі елементи:

– crowdfunding – компонент збору;

– створення користувачем збору – компонент за допомогою якого користувачі можуть створювати збори;

– підтвердження адміністратором – компонент за допомогою якого збір відображається всім користувачам і виконується смарт-контракт для створення збору;

– смарт-контракт для створення зборів – компонент за допомогою якого відбувається взаємодія з blockchain;

– підтримка коштами автора збору – проведення спонсорами переказу коштів на гаманець автора збору за допомогою технології blockchain.

Висновки до другого розділу

В процесі проектування платформи краудфандингу були проаналізовані усі варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована

структура платформи. Проектування включало в себе моделювання структури проекту та взаємодії між їх складовими частинами.

Також було спроектовано базу даних для зберігання значень перед потраплянням їх на блокчейн, та для даних які будуть зберігатися на серверній частині додатку, таких як користувачі та категорії

Було описано алгоритм роботи з додатком та описані послідовні пункти варіантів його використання.

Також був наведений опис основного функціоналу додатку – це створення збору користувачем, його підтвердження адміністрацією та публікація його в блокчейні, також були розглянуті основні функції по взаємодії користувача зі збором

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КРАУДФАНДИНГОВОЮ ПЛАТФОРМОЮ

3.1 Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування платформи вимагає розгортання всіх його компонентів та налаштування як кожного елементу окремо, так і загальної системи в цілому.

Для опису архітектури платформи було створено діаграму розгортання (рис. 3.1).

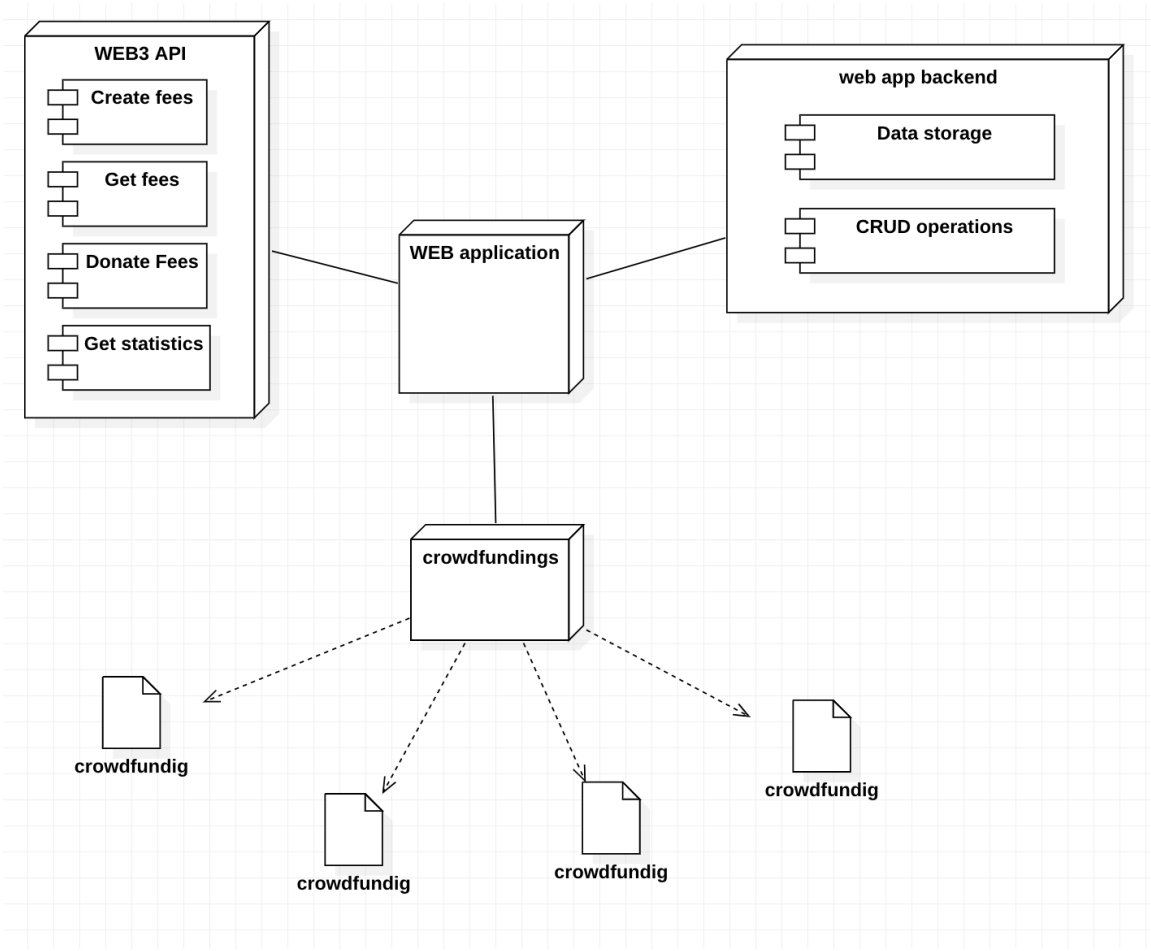

Рис. 3.1. Діаграма розгортання платформи краудфандингу

Опис процес розгортання.

WEB3 API – розгортання смарт-контракту в мережі Ethereum для взаємодії з блокчейном за допомогою методів смарт-контракту.

Create fees – метод смарт-контракту, що дозволяє користувачам строювати збір.

Get fees – метод смарт-контракту, що дозволяє користувачам отримувати збори.

Donate fees – метод смарт-контракту, що дозволяє здійснювати переказ коштів автору збору.

Get statistics – метод смарт-контракту, що дозволяє отримати данні для відображення статистики.

WEB APP BACKEND – розгортання серверної частини додатку, для зберігання проміжних даних та проведення CRUD операцій з даними, які не зберігаються в блокчейні.

WEB APP FRONTENTD – розгортання клієнтської частини додатку яка об’єднує данні з серверної частини на блокчейну, також відповідає за візуалізацію цих даних та обміном даних між серверною частиною та блокчейном.

3.2 Структура інтерфейсу краудфандингової платформи

Для використання вебдодатку користувачам потрібно встановити розширення MetaMask [24].

MetaMask - це розширення для браузера, який дозволяє користувачам взаємодіяти з блокчейнами, зокрема з Ethereum, і використовувати смарт-контракти. Він виступає як універсальний гаманець та інтерфейс для взаємодії з децентралізованими додатками (DApps) на Ethereum.

Основна роль MetaMask полягає в забезпеченні сполучення між веббраузером користувача та Ethereum-мережею. Він дозволяє користувачам створювати та керувати своїми Ethereum-адресами, зберігати приватні ключі і підписувати транзакції. За допомогою MetaMask користувачі можуть взаємодіяти зі смарт-контрактами, відправляти та отримувати криптовалюту, виконувати функції контрактів та багато іншого.

MetaMask також надає зручний інтерфейс для авторизації та ідентифікації користувачів на різних додатках та вебсайтах, що побудовані на Ethereum. Він

дозволяє користувачам підтверджувати свою участь у транзакціях та виконувати дії на блокчейні, використовуючи свої приватні ключі та підписи.

У контексті смарт-контрактів, MetaMask дозволяє користувачам інтерактивно взаємодіяти з контрактами, виконувати функції, відправляти транзакції та перевіряти стан контракту. Користувачі можуть ініціювати транзакції на MetaMask, підтверджувати їх та взаємодіяти з інтерфейсом контракту, що дозволяє взаємодіяти з децентралізованими додатками та виконувати різноманітні операції на Ethereum.

MetaMask також надає розширений функціонал для розробників смарт-контрактів. Він надає набір API, які дозволяють взаємодіяти з MetaMask з додатків та вебсайтів. Розробники можуть використовувати ці API для отримання інформації про аккаунти користувачів, стан гаманців, відправки транзакцій та багато іншого.

MetaMask також забезпечує безпеку під час взаємодії з смарт-контрактами. Користувачі можуть перевіряти та підтверджувати деталі транзакцій перед їх відправкою, що дозволяє уникнути непередбачених витрат коштів чи вразливостей у контрактах (рис.3.2).

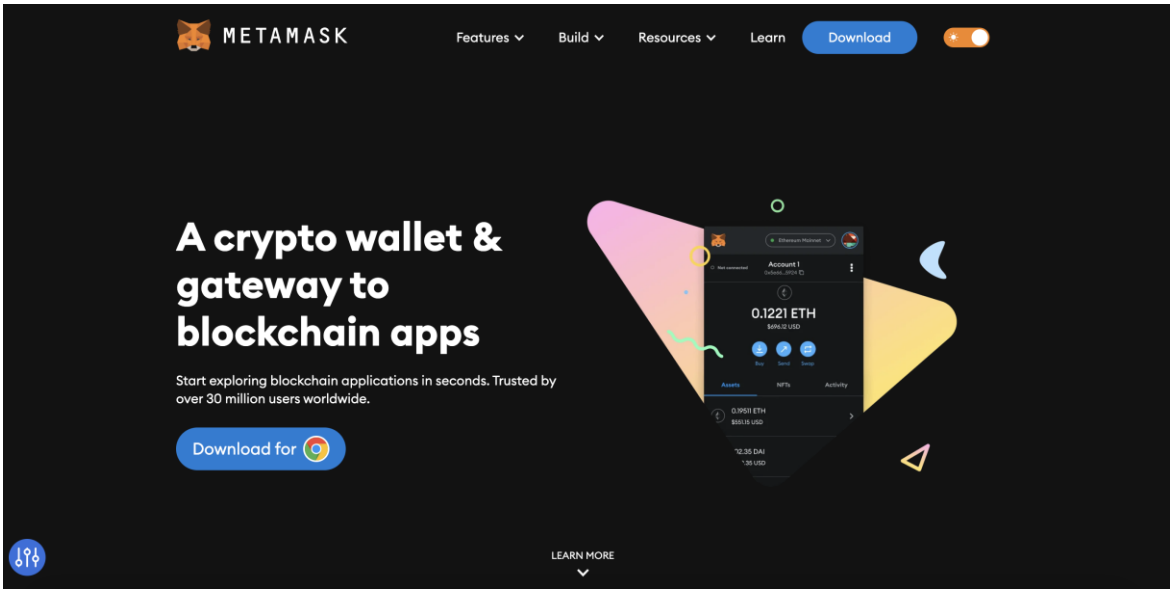

Рис. 3.2. Головна сторінка MetaMask

Узагалі, MetaMask грає важливу роль у полегшенні взаємодії користувачів з Ethereum та смарт-контрактами. Він дозволяє звичайним користувачам легко

використовувати децентралізовані додатки та взаємодіяти з екосистемою Ethereum, необхідним для виконання різних операцій на блокчейні.

Взаємодія користувачів відбувається через вебдодаток. На головній сторінці користувач може побачити останні три створені збори та останні три коментаря які залишили користувачі або власники зборів (рис 3.3 – рис. 3.4).

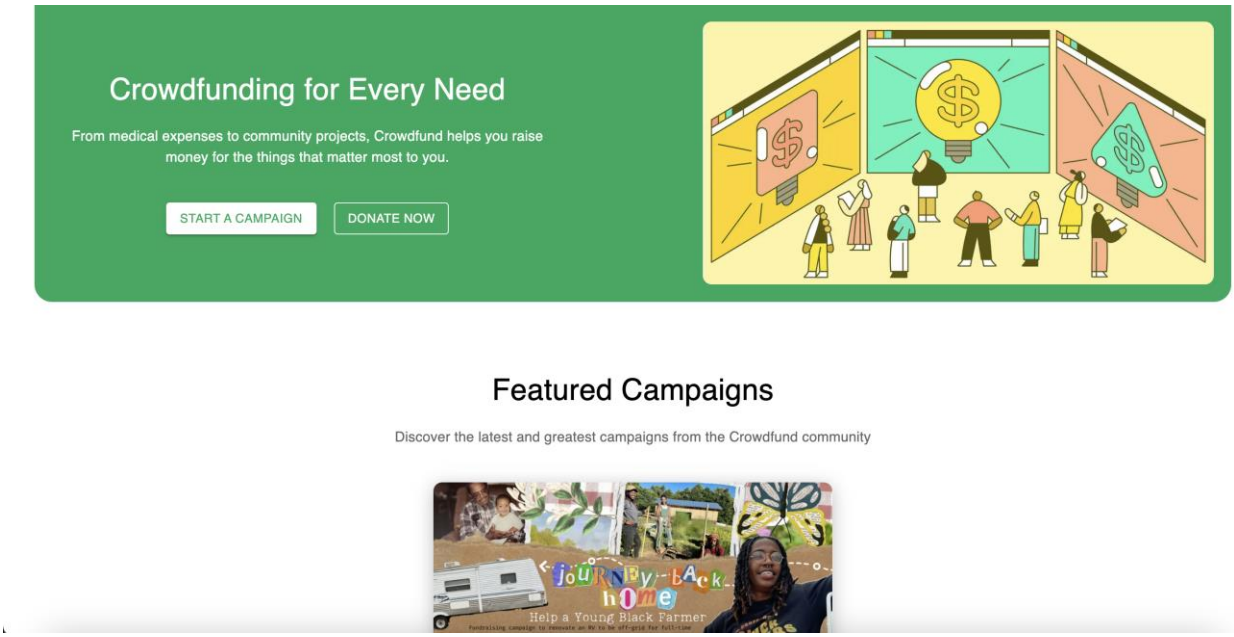

Рис. 3.3. Головна сторінка додатку

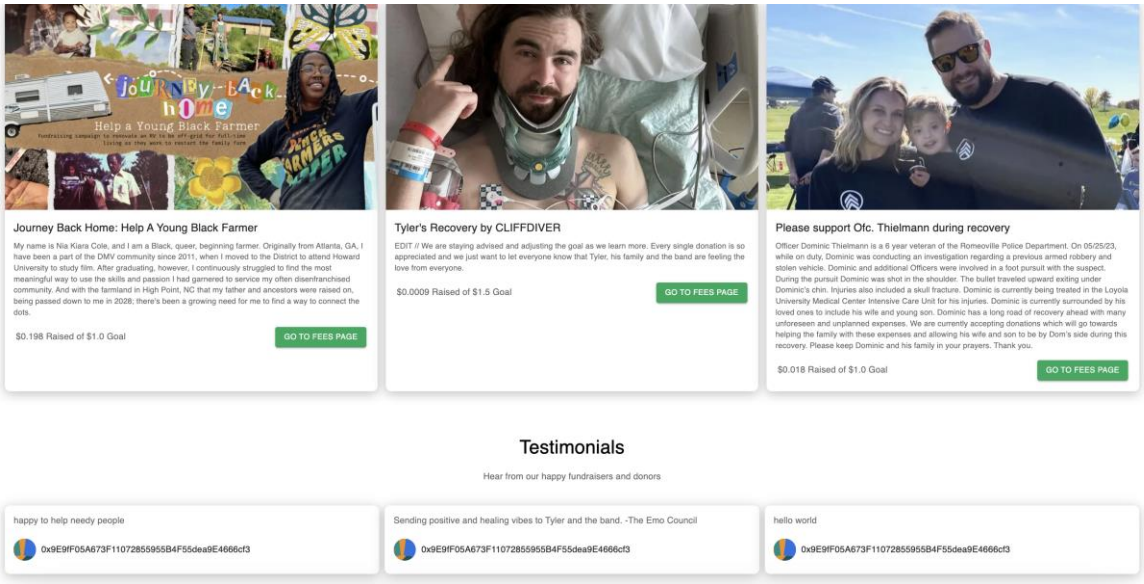

Рис. 3.4. Головна сторінка додатку

Користувач має можливість перейти до зборів, або створити профіль де в подальшому може створити власний збір. Розглянемо роль користувача який зайшов підтримати збір.

Користувач на сторінці зборів може переглянути всі збори відфільтрувати її по категоріям, або відсортувати по різним параметрам (рис 3.5).

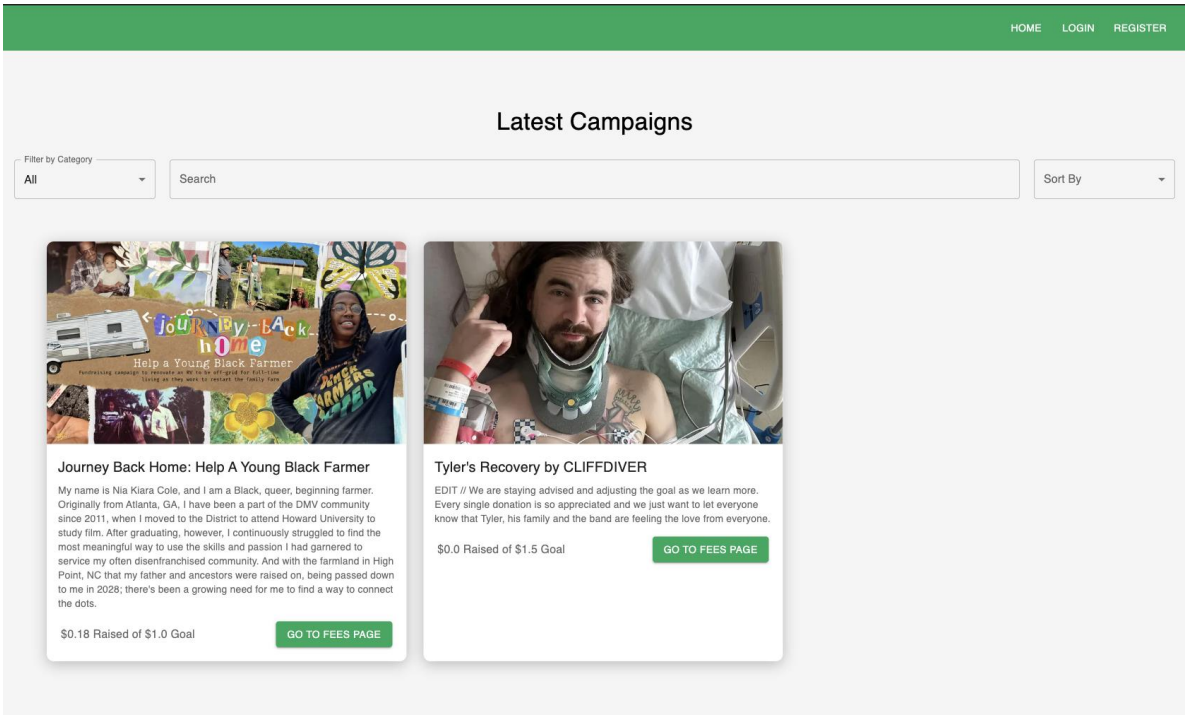

Рис. 3.5. Сторінка зборів

Після обрання збору користувач може перейти на сторінку збору де може переглянути більш детальну інформацію про збір (рис 3.6).

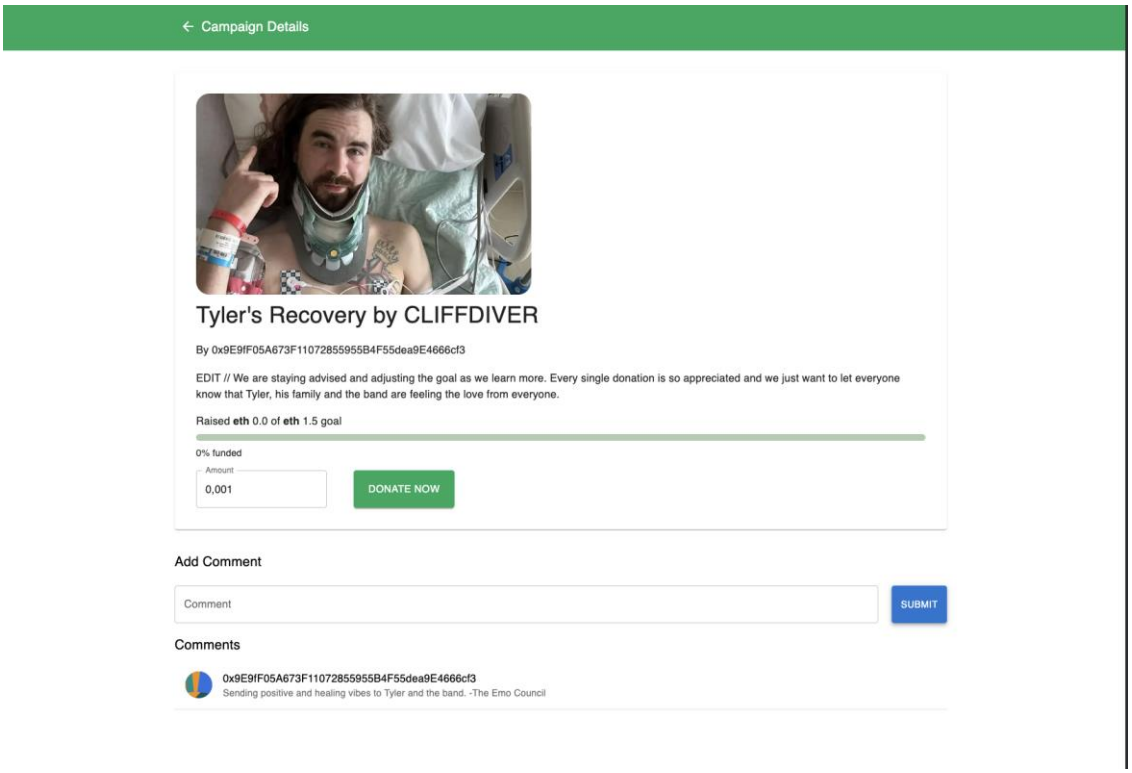

Рис. 3.6. Сторінка збору

Тут у користувача є можливість здійснити переказ коштів на збір або залишити коментар, для цього йому потрібно авторизуватись за допомогою MetaMask.

При переказі коштів у користувача відкриється вікно де йому потрібно підтвердити переказ коштів (рис 3.7).

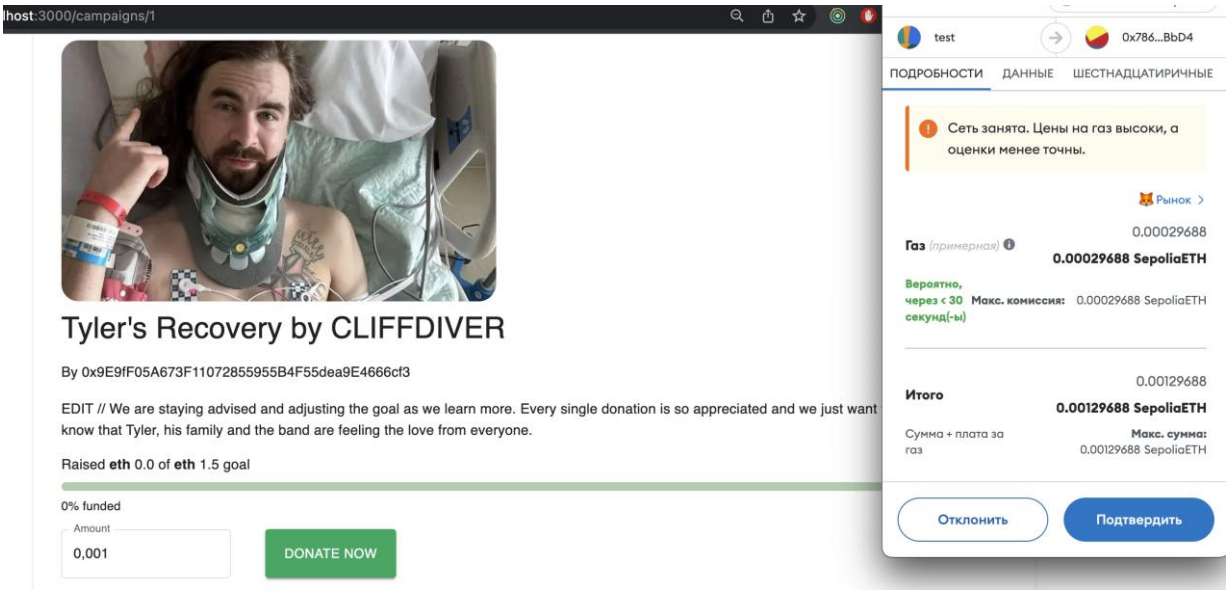

Рис. 3.7. Переказ коштів

Також користувач може залишити коментар під збором (рис. 3.8 – рис .3.9).

Рис. 3.8 Створення коментаря

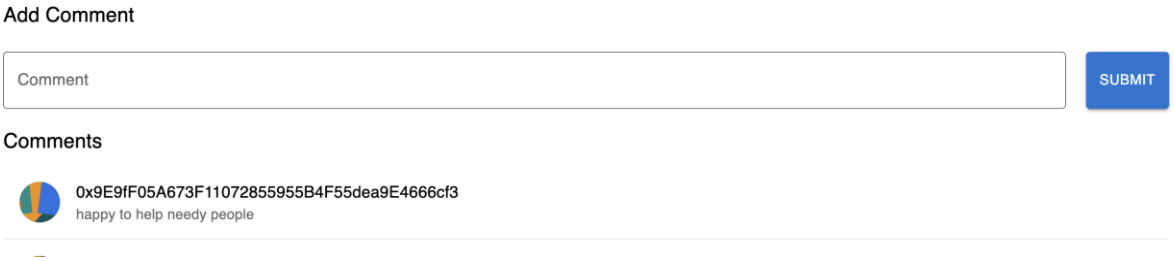

Рис. 3.9. Відображення коментарів

Розглянемо процес створення збору користувачем. Для цього потрібно зареєструватись в додатку, або авторизуватись, якщо користувач вже існує (рис. 3.10 – рис. 3.11.)

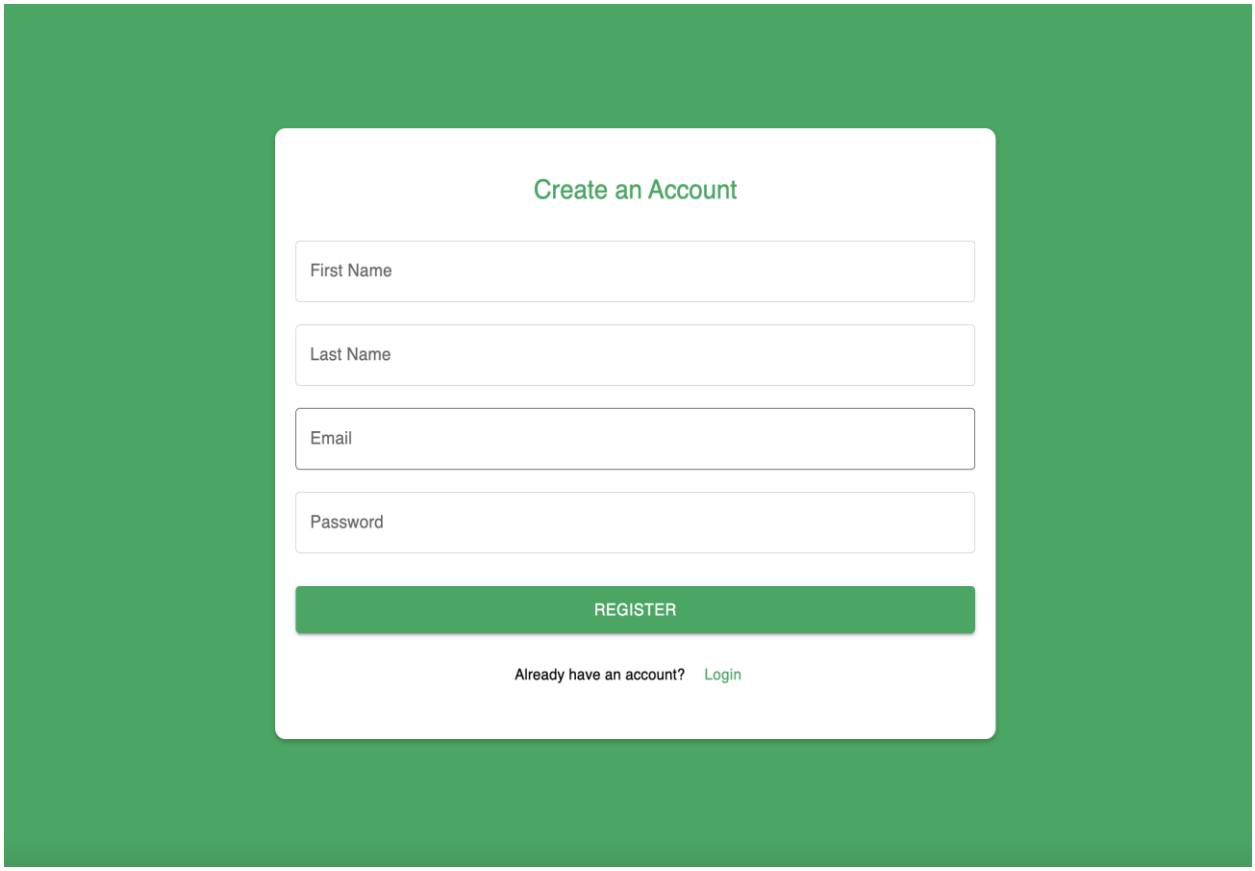

Рис. 3.10 Сторінка реєстрації

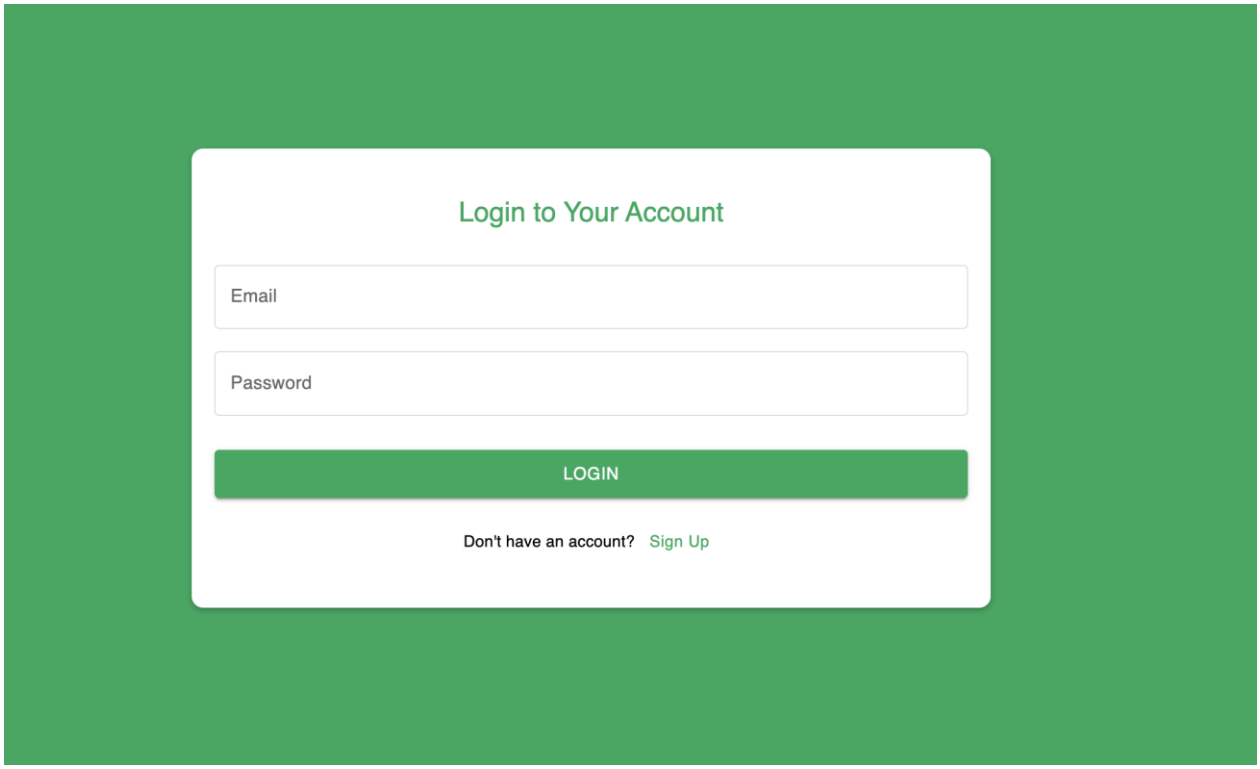

Рис. 3.11. Сторінка авторизації

Після авторизації користувач може створити збір (рис 3.12).

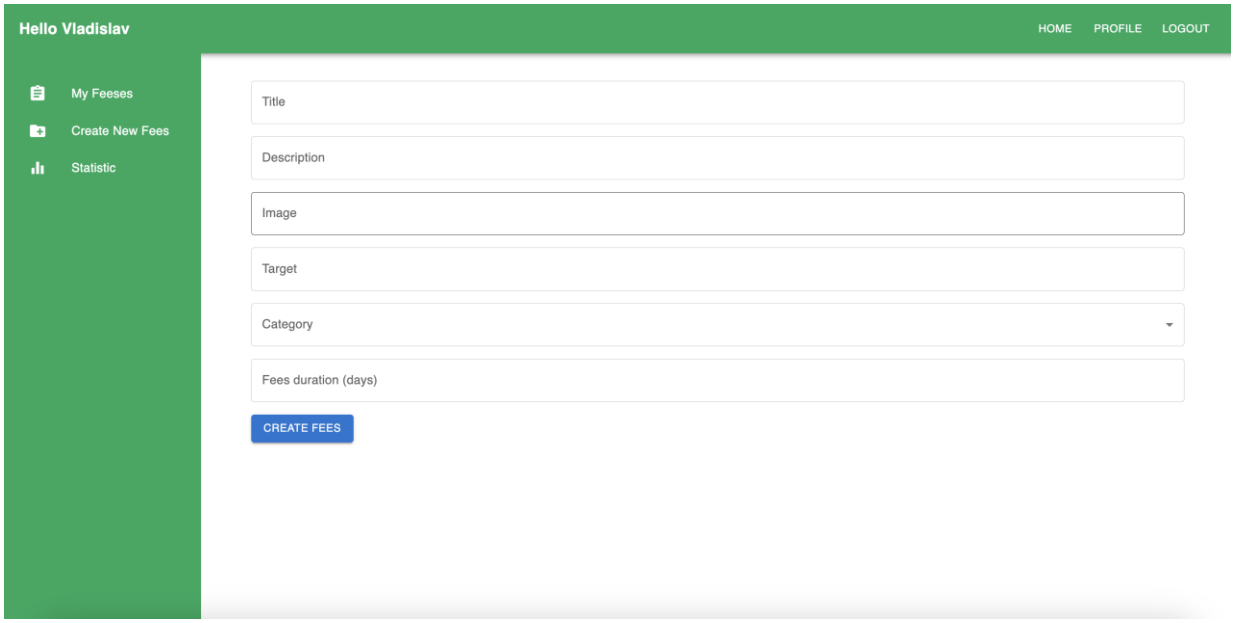

Рис. 3.12. Створення збору користувачем

Після створення збору він потрапляє на перегляд адміністрацією, де має бути підтверджений, оскільки користувачі можуть створити збір який не задовольняє принципи сайту або порушує чинне законодавство, тому валідація адміністрацією важливий етап створення збору (рис 3.13).

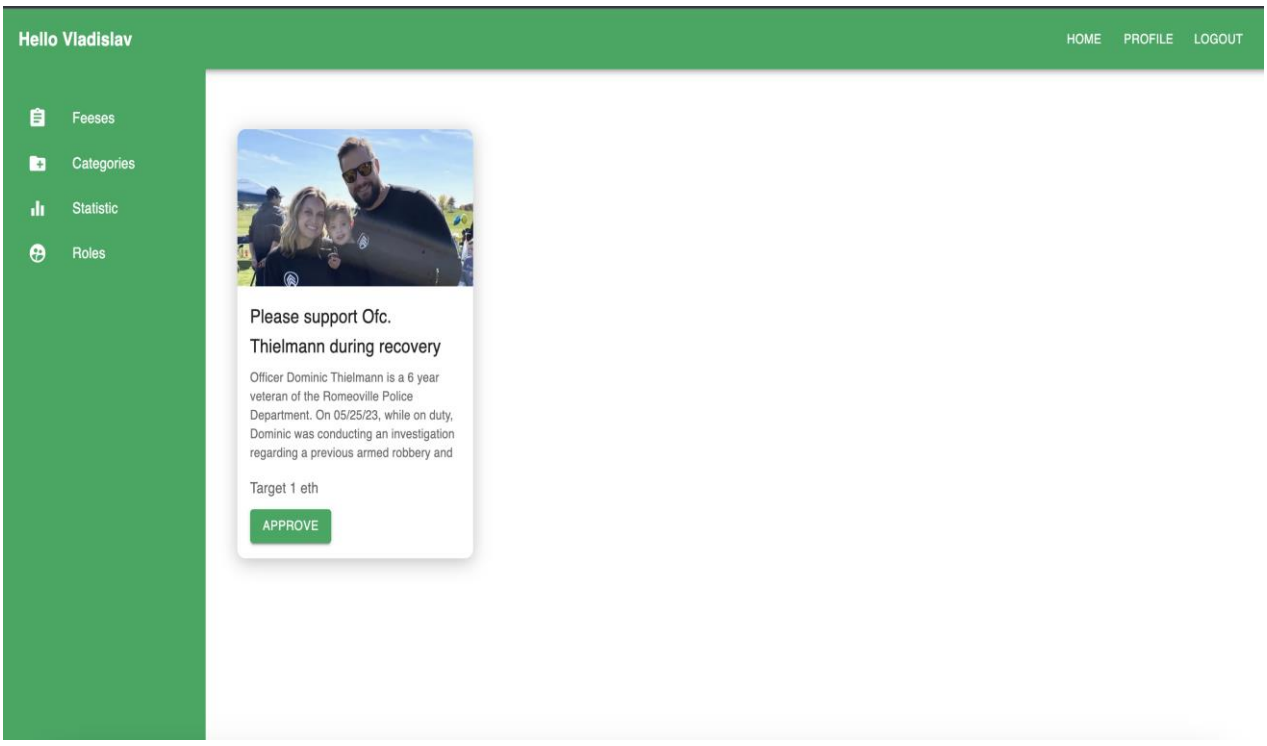

Рис. 3.13. Підтвердження збору адміністрацією

Після підтвердження адміністрацією збір стає доступним для публікації. Під час цього процесу відбувається взаємодія зі смарт контрактом який має бути

підтверджений користувачем аналогічно до переказу коштів на збір, після чого він потрапляє в мережу (рис. 3.14 – рис. 3.15).

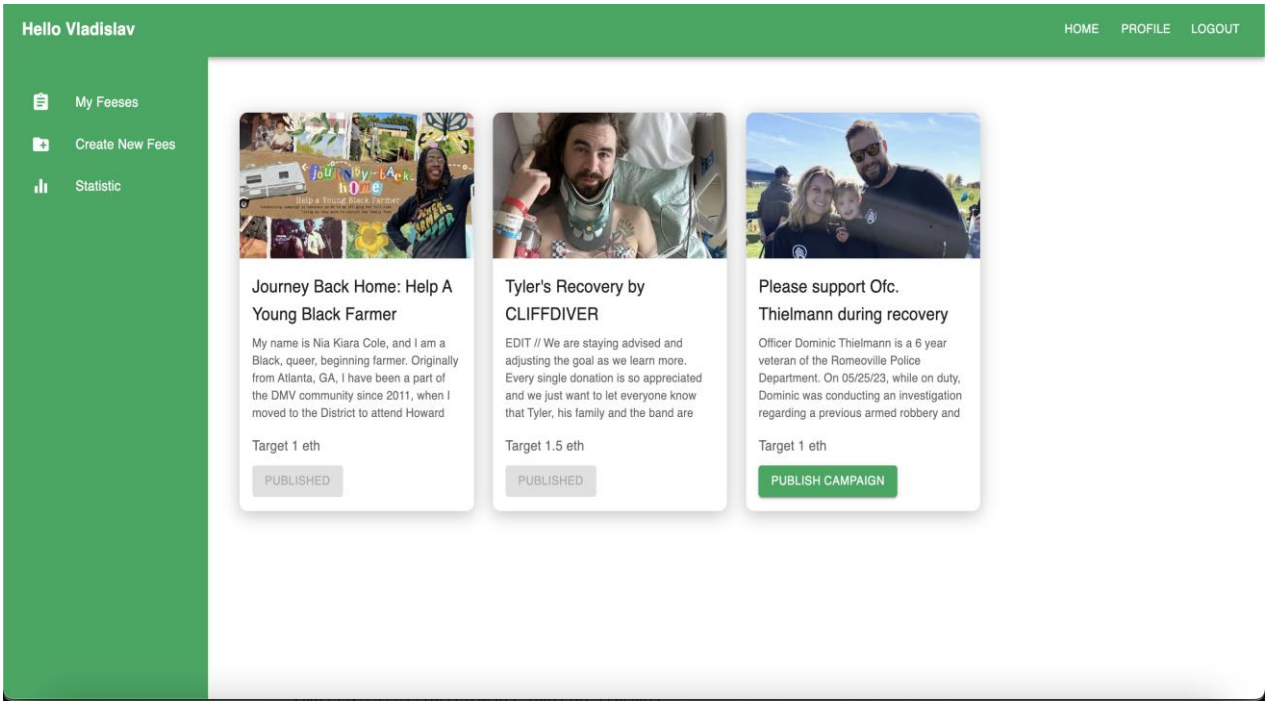

Рис. 3.14. Публікація збору

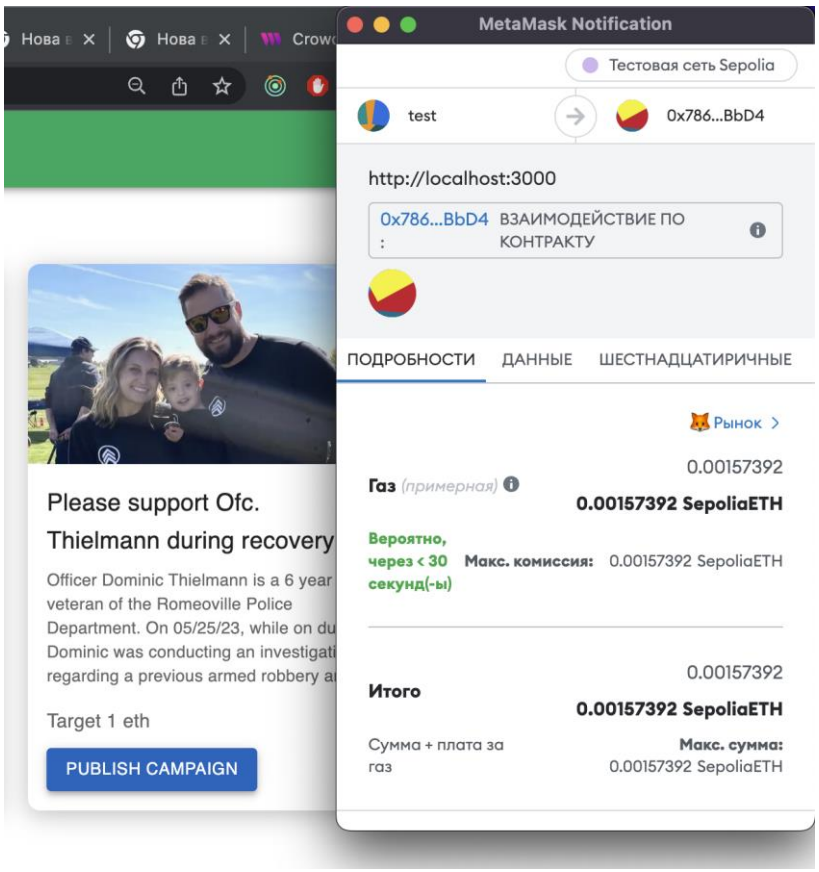

Рис. 3.15. Публікація збору

Після публікації кнопка стає неактивною (рис 3.16).

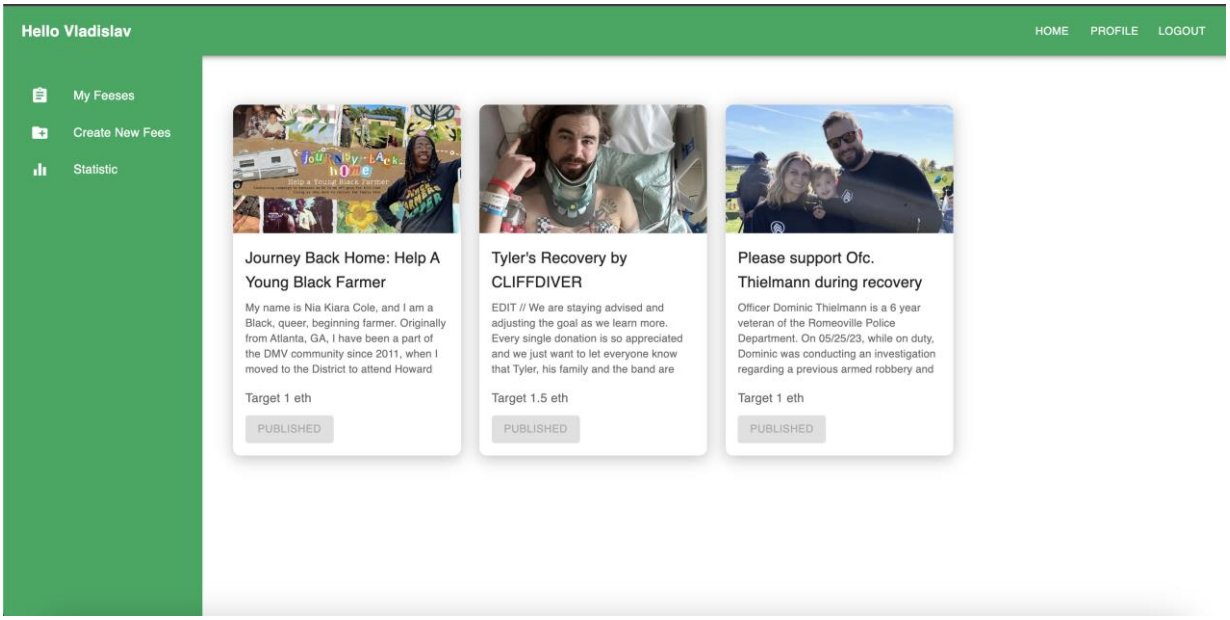

Рис. 3.16 Опубліковані збори

У адміністратора також є можливість створювати нові категорії, або редагувати їх (рис. 3.17 – рис. 3.19)

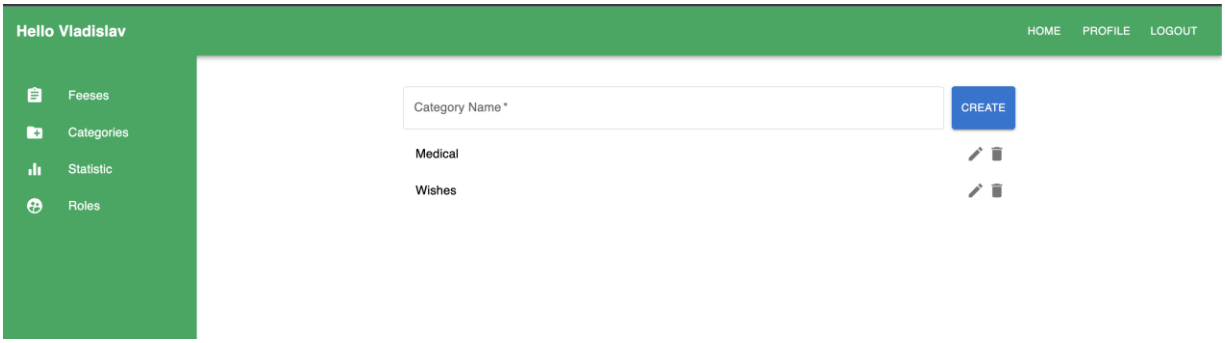

Рис. 3.17. Редактор категорій

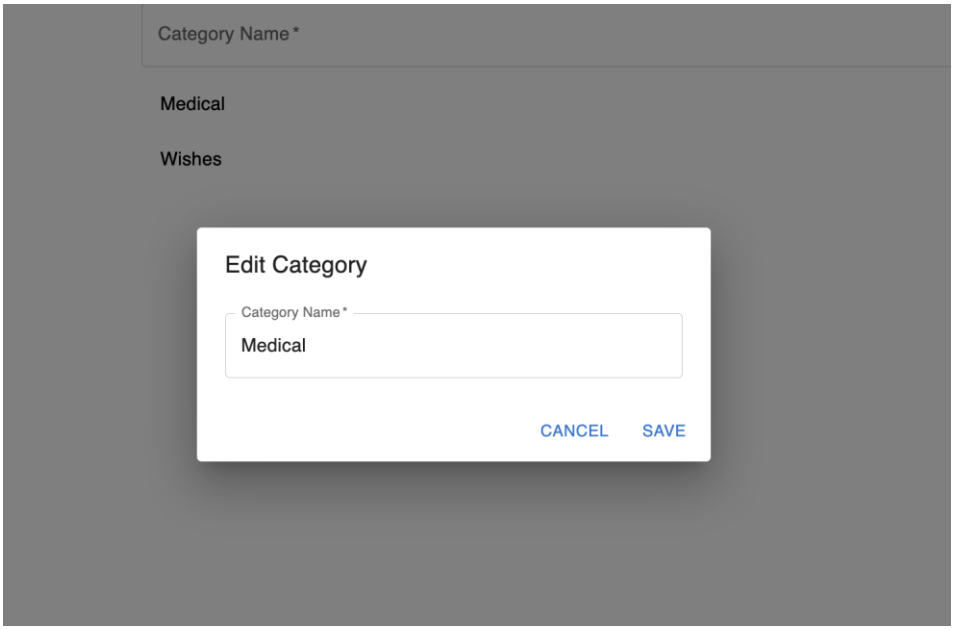

Рис. 3.18. Редагування категорії

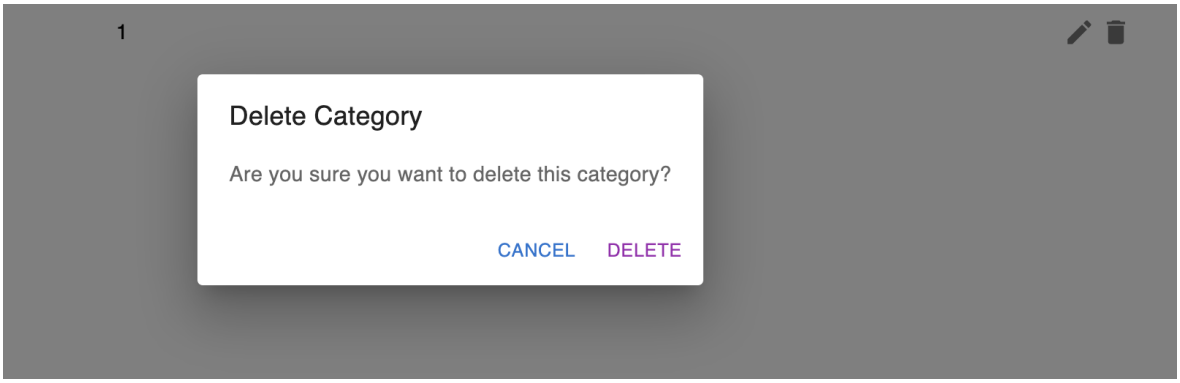

Рис. 3.19. Видалення категорії

Також адміністратор має можливість редагувати ролі користувачів, якщо потрібно додати нового адміністратора (рис. 3.20 – рис. 3.21).

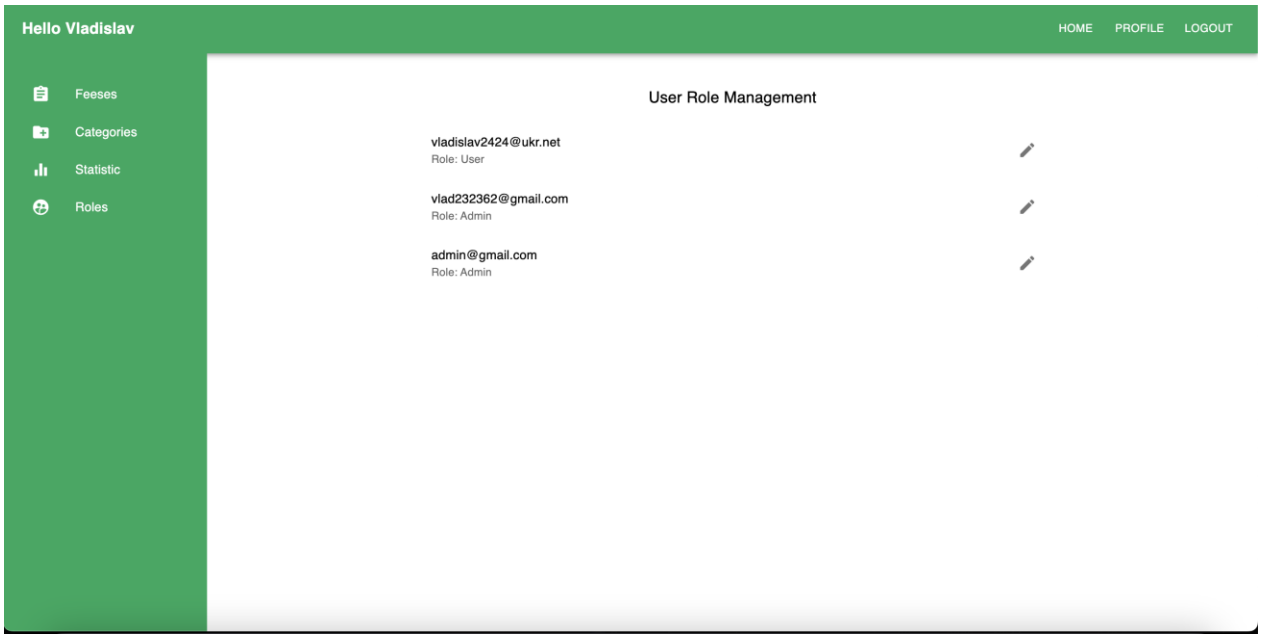

Рис. 3.20. Менеджер ролей

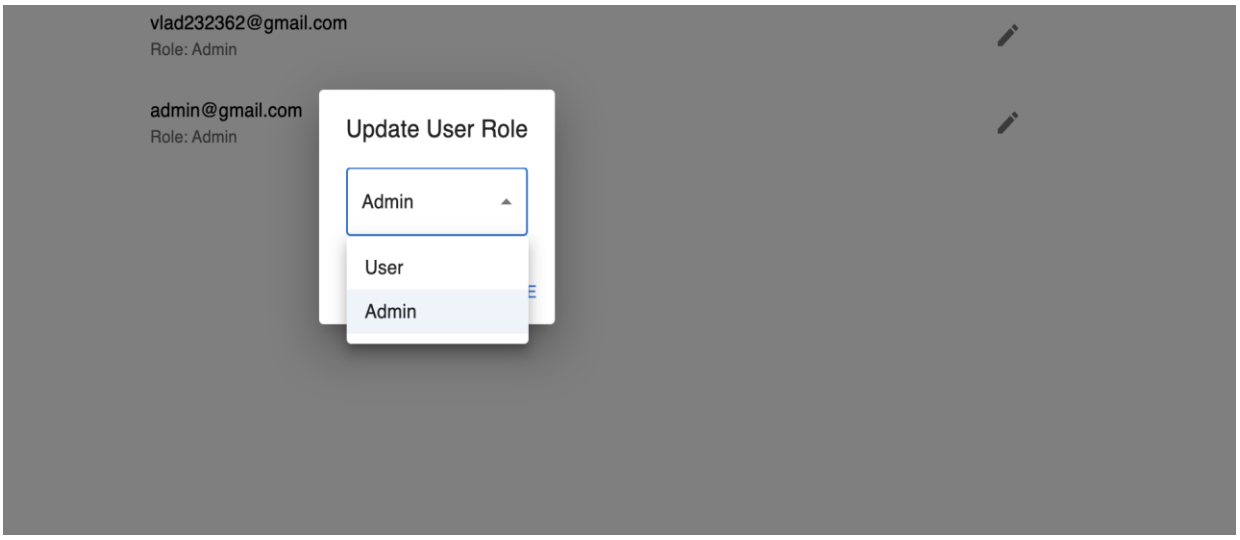

Рис. 3.21. Зміна ролі користувача

3.3 Статистика зборів та загальна статистика платформи

У користувачів та адміністрації є можливість переглянути статистику по зборам, це дозволяє відслідковувати динаміку зборів. Адміністрація має доступ до всіх зборів, тому в них відображається загальна статистика по зборам, та скільки коштів отримала платформа, скільки коштів було зароблено, скільки всього було здійснено донатів та скільки всього зборів створено. Також на діаграмах зображено загальну статистику по категоріям та конкретним зборам (рис. 3.22 – рис. 3.23).

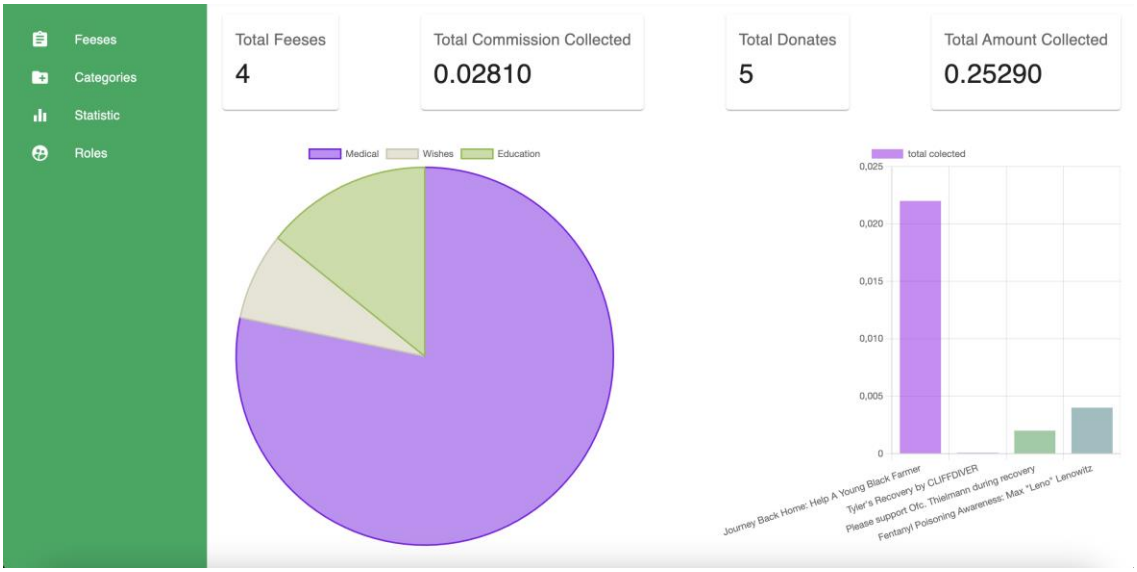

Рис. 3.22. Статистика адміністратора

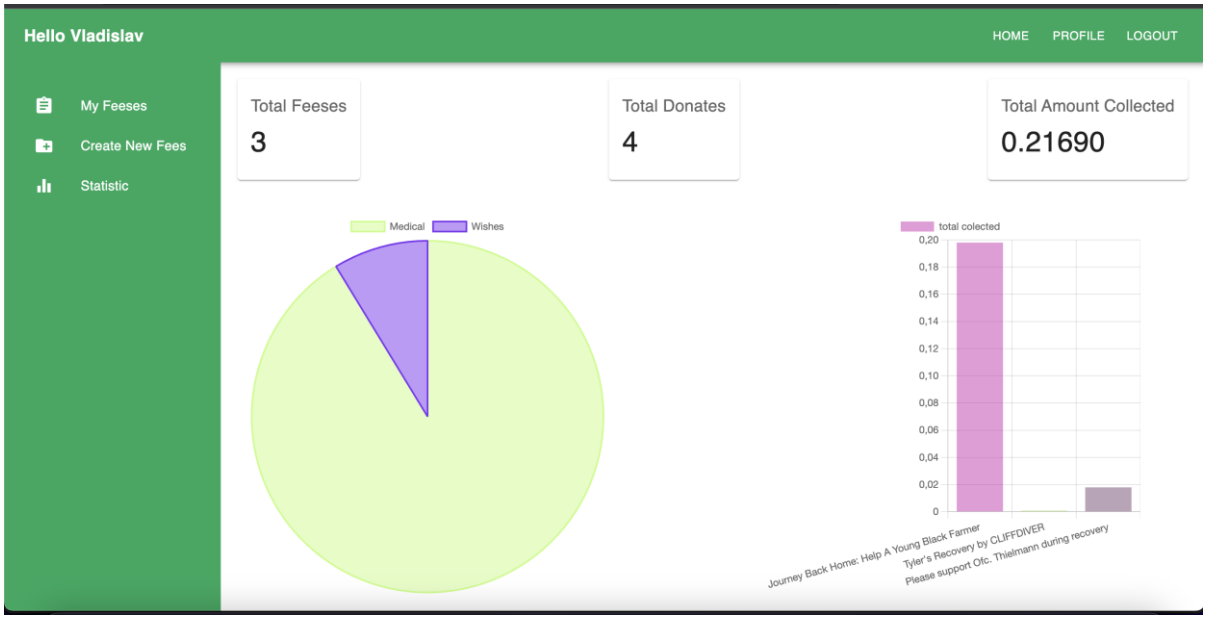

Рис. 3.23. Статистика користувача

У користувача в діаграмах відображається сума яка була отримана, у адміністратора комісія платформи, тобто сума яку заробила платформа, також оскільки користувач може створювати збори використовуючи різні гаманці відображається статистика активного гаманця, при бажанні користувач може змінити гаманець в MetaMask і йому буде відображена статистика по відповідному гаманцю.

Якщо статистики по відповідному гаманцю не було знайдено буде відображено відповідне повідомлення (рис. 3.24).

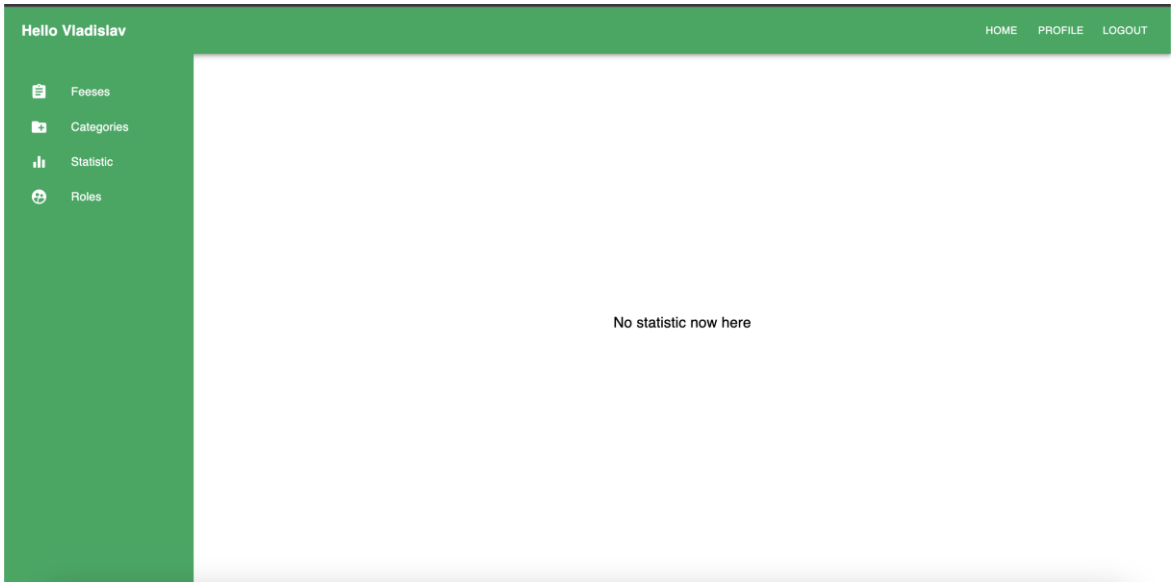

Рис. 3.24. Повідомлення про відсутність статистики

3.4 Тестування платформи

За допомогою онлайн сервісу було проведено тестування вебдодатку. Для тестування було використано GTMetrix та Pagespeed.

GTMetrix – це інструмент для тестування продуктивності та швидкості завантаження вебсторінок. Він надає звіти та аналіз продуктивності вашого вебсайту на основі різних метрик, таких як час завантаження сторінки, розмір сторінки, кількість запитів до сервера і багато іншого [25].

Після завершення тестування було отримано звіт, який містить різноманітну інформацію про продуктивність сайту (рис. 3.25).

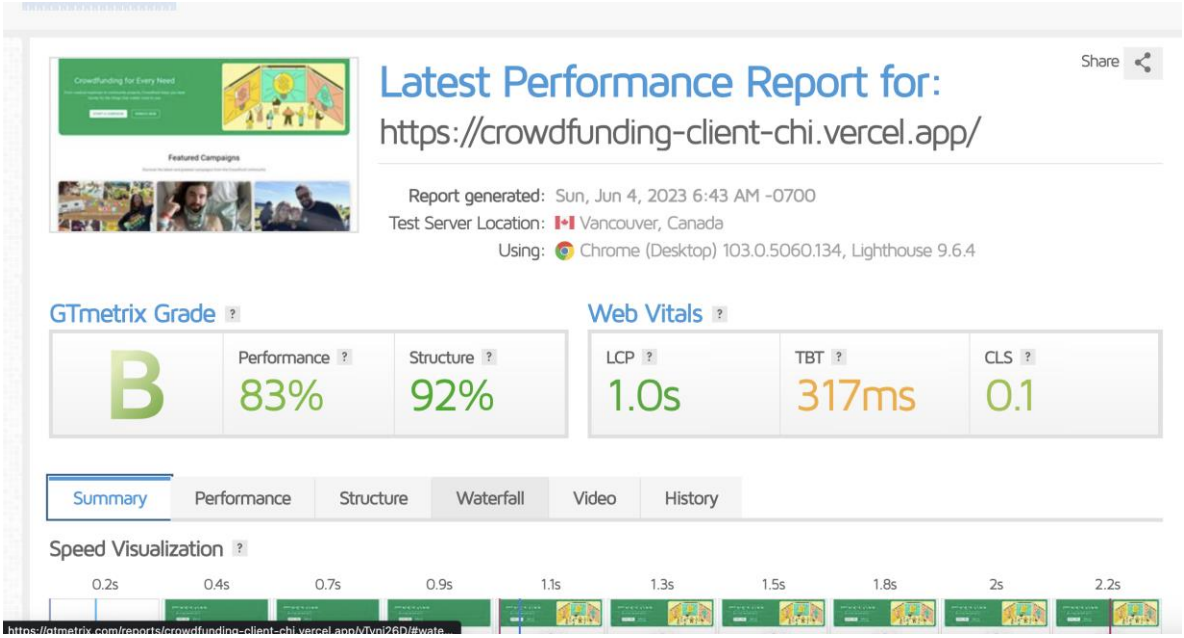

Рис. 3.25. Тестування вебдодатку за допомогою GTMetrix

Звіти GTMetrix містять такі дані, як загальний час завантаження сторінки, час завантаження вмісту, кількість запитів до сервера, розмір сторінки, швидкість завантаження на різних типах з'єднань і багато іншого. Ця інформація допомагає вам зрозуміти, як ваш вебсайт працює з точки зору продуктивності та швидкості, і дозволяє вам вжити заходів для поліпшення цих показників (рис.2.26).

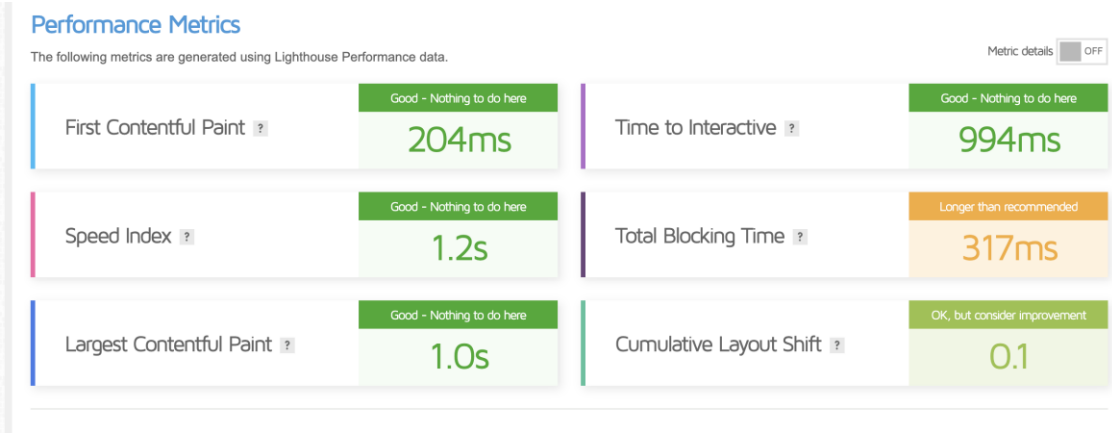

Рис. 3.26. Тестування швидкості вебдодатку за допомогою GTMetrix

GTMetrix також надає ряд інструментів та функцій для оптимізації продуктивності вебсайту, включаючи можливість зберігання та порівняння звітів, моніторинг продуктивності, сповіщення про проблеми продуктивності та багато іншого.

Pagespeed (рис. 3.27) вебсайт, який надає інструменти та рекомендації щодо оптимізації швидкості завантаження сторінок. Цей вебсайт створений командою Google з метою допомогти веброзробникам покращити продуктивність своїх вебсайтів [26].

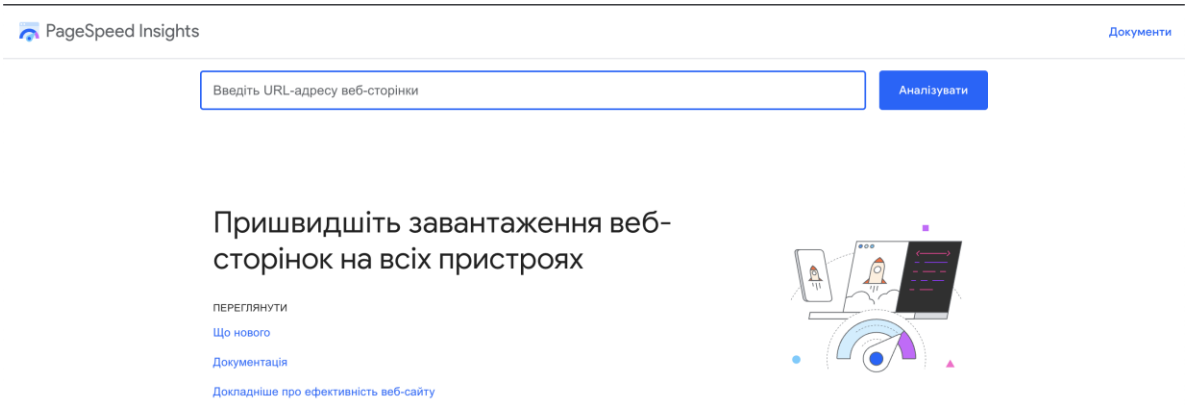

Рис. 3.27. Веб сайт Pagespeed

Аналіз швидкості сторінок. Вебсайт надає можливість ввести URL-адресу свого вебсайту і отримати докладний аналіз швидкості його завантаження. Інструмент оцінює різні аспекти, включаючи час завантаження, розмір сторінки, кількість запитів до сервера та інші параметри (рис. 3.28).

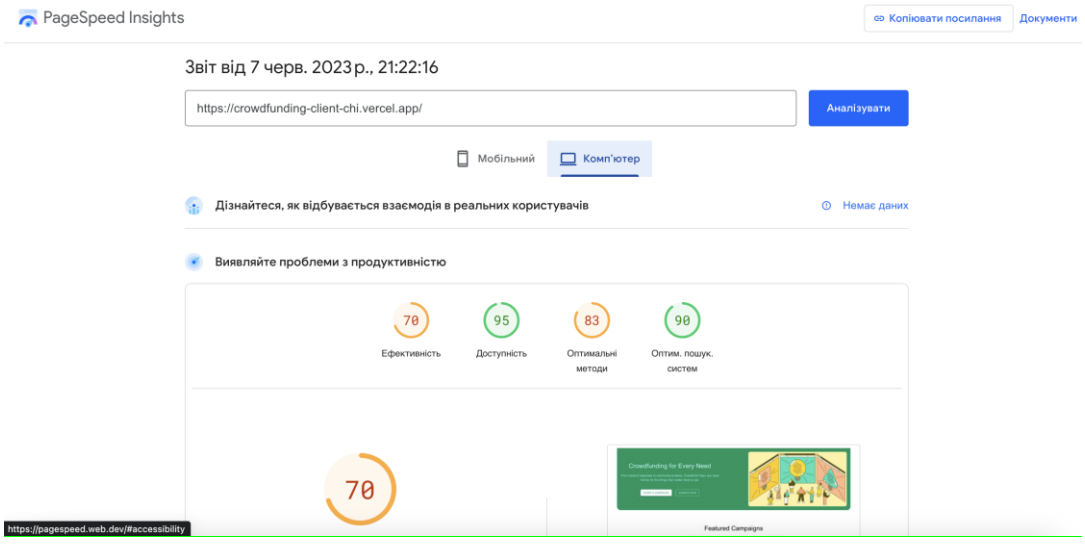

Рис. 3.28. Тестування вебдодатку за допомогою Pagespeed

Рекомендації щодо оптимізації. Вебсайт надає докладний список рекомендацій та практичні поради щодо того, як покращити швидкість завантаження сторінок. Ці рекомендації базуються на кращих практиках та рекомендаціях Google для оптимізації вебсайтів.

Перевірка на мобільну готовність. Окрім аналізу швидкості сторінок, вебсайт також перевіряє готовність вебсайту до мобільних пристроїв. Він надає рекомендації щодо адаптивного дизайну, розміщення контенту та інших аспектів, які можуть впливати на використання вебсайту на мобільних пристроях (рис. 3.29).

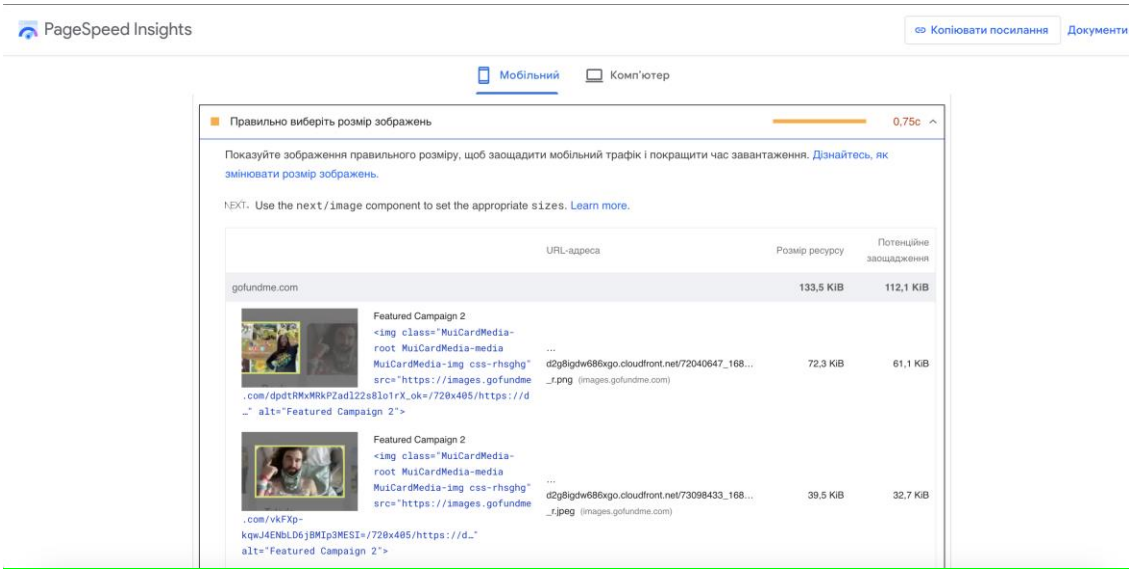

Рис. 3.29. Тестування веб додатку за допомогою Pagespeed

Перегляд різних метрик. Вебсайт надає можливість переглядати різні метрики швидкості, такі як Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), Total Blocking Time (TBT) та інші. Це допомагає зосередитися на конкретних показниках і виправити проблеми, що впливають на продуктивність (рис. 3.30).

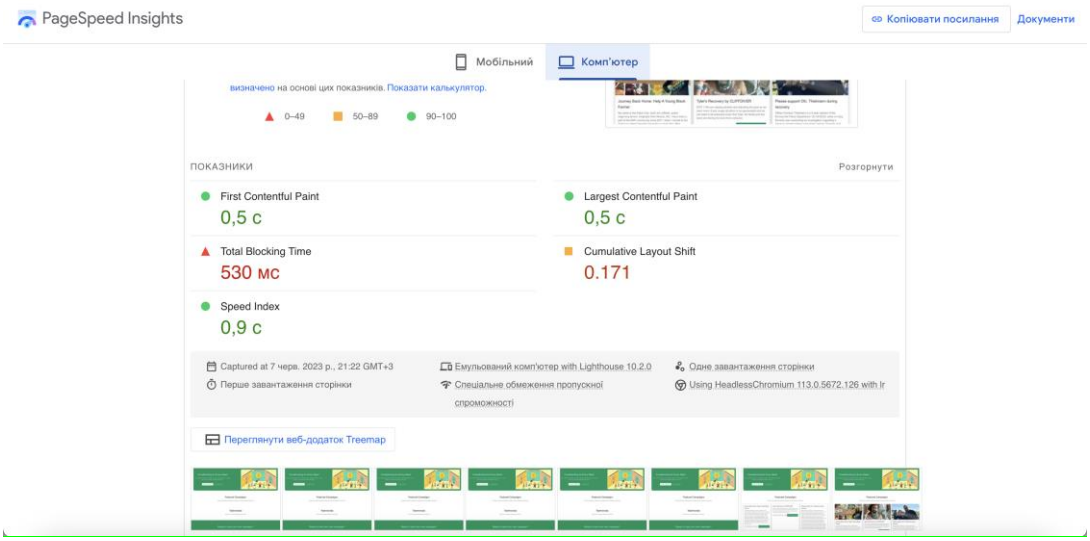

Рис. 3.30. Тестування вебдодатку за допомогою Pagespeed

Рекомендації щодо вдосконалення. Вебсайт надає конкретні поради та рекомендації щодо вдосконалення швидкості завантаження сторінок. Це можуть бути вказівки щодо стиснення зображень, кешування, мінімізації CSS та JavaScript файлів, використання асинхронного завантаження ресурсів та інші оптимізаційні техніки (рис.3.31).

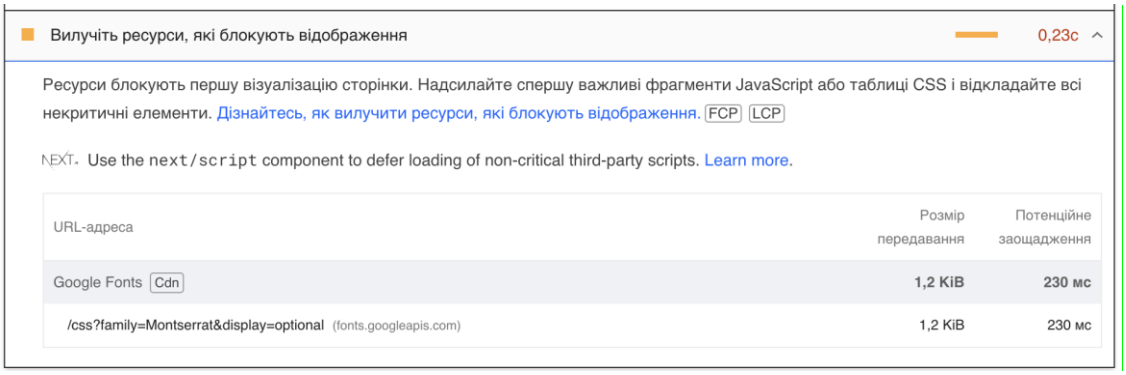

Рис. 3.31. Тестування вебдодатку за допомогою Pagespeed

Підтримка для різних платформ. Вебсайт пропонує поради та рекомендації для різних платформ та технологій, включаючи вебсайти на основі WordPress, Shopify, Angular, React та інші. Це дозволяє розробникам вибрати відповідні стратегії оптимізації для їх конкретної платформи.

Лабораторні показники. Вебсайт надає детальну інформацію про різні показники швидкості та продуктивності, такі як перше відображення, час інтерактивності, час завантаження повного вмісту та багато інших. Це допомагає вам зрозуміти, як кожен аспект впливає на швидкість завантаження сторінок і як їх можна оптимізувати.

Документація та ресурси. Вебсайт надає багато корисної документації, статей, керівництв та ресурсів, які допомагають зрозуміти принципи оптимізації швидкості завантаження сторінок. Можна дізнатись більше про різні аспекти та стратегії для покращення продуктивності вашого вебсайту (рис. 3.32).

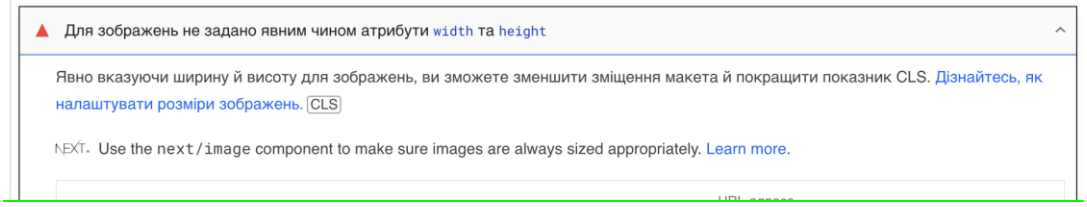

Рис. 3.32. Тестування вебдодатку за допомогою Pagespeed

За допомогою інструментів GTMetrix та Pagespeed було проведено тестування продуктивності та швидкості завантаження вебдодатку. За результатами тестування отримано метрики, такі як час завантаження сторінки, розмір сторінки, кількість запитів до сервера, Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), Total Blocking Time (TBT) та інші показники.

Висновки до третього розділу

В даному розділі була наведена детальна схема розгортання платформи. Кожен структурний елемент системи був описаний, а також було продемонстровано його взаємодію з іншими частинами платформи. Це дозволило зрозуміти архітектуру системи та взаємозв'язки між її компонентами.

Далі було надано опис інтерфейсу розробленого програмного продукту. Були розглянуті всі функціональні можливості додатку, залежно від ролі користувача. Кожен тип користувача отримав свої унікальні функції та можливості в системі. Це допомагає кожному користувачу зорієнтуватись у функціоналі додатку та використовувати його відповідно до своїх потреб.

Окрім того, було продемонстровано, як працює статистика для адміністрації та звичайних користувачів. Була показана різниця між ними та продемонстровано, що конкретно відображають різні діаграми та статистичні дані. Адміністраторам надається детальна інформація про статус збору коштів, активність донорів та статистику за категоріями.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано платформу краудфандингу, проведено аналіз предметної області та порівняно різні платформи, що працюють на основі збору коштів або підтримки авторів. На основі цього аналізу були визначені основні вимоги до платформи краудфандингу. В процесі дослідження була розроблена архітектура, підходи та основні технології реалізації вебдодатку.

Проектування платформи включало аналіз варіантів використання програмного продукту та проектування структури, включаючи моделювання взаємодії між складовими частинами. Також було спроектовано базу даних для зберігання інформації перед її записом на блокчейн, а також для даних, що зберігаються на серверній частині додатку.

Описано алгоритм роботи з додатком та послідовні пункти варіантів його використання. Розглянуто основний функціонал, включаючи створення збору, підтвердження адміністрацією та публікацію в блокчейні, а також основні функції взаємодії користувачів зі збором.

Також була надана детальна схема розгортання платформи краудфандингу, описано взаємодію структурних елементів системи та наведено опис інтерфейсу розробленого програмного продукту для різних типів користувачів. Також показано роботу статистики для адміністрації та звичайних користувачів, що надає інформацію про статус збору коштів, активність донорів та статистику за категоріями.

Архітектура платформи побудована на базі блокчейну, що гарантує прозорість та недоторканність зібраних коштів. Смарт-контракти використовуються для автоматизації процесів збору та розподілу коштів, забезпечуючи довіру між учасниками платформи.

Додаток має інтуїтивний інтерфейс, який дозволяє користувачам легко створювати збори, встановлювати мету фінансування та опис проекту. Користувачі можуть взаємодіяти зі зборами, приєднуватися до них, вносити внески та спостерігати за ходом збору коштів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gofundme [Електронний ресурс] – Режим доступу до ресурсу: https://www.gofundme.com/
2. Kickstarter [Електронний ресурс] – Режим доступу до ресурсу: https://www.kickstarter.com/?ref=nav
3. Indiegogo [Електронний ресурс] – Режим доступу до ресурсу: https://www.indiegogo.com/.
4. Crowdfunder [Електронний ресурс] – Режим доступу до ресурсу: https://www.gofundme.com/
5. Patreon [Електронний ресурс] – Режим доступу до ресурсу: https://www.patreon.com/UA
6. NestJS [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nestjs.com/
7. NextJS [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/
8. NextJS SSR [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering
9. Solidity [Електронний ресурс] – Режим доступу до ресурсу: https://soliditylang.org/
10. Solidity [Електронний ресурс] – Режим доступу до ресурсу: https://docs.soliditylang.org/en/develop/
11. Solidity [Електронний ресурс] – Режим доступу до ресурсу: https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
12. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/
13. ThirdWeb [Електронний ресурс] - Режим доступу до ресурсу: https://thirdweb.com/

14. useAddress [Електронний ресурс] - Режим доступу до ресурсу: https://portal.thirdweb.com/react/react.useaddress

15. useContract [Електронний ресурс] - Режим доступу до ресурсу: https://portal.thirdweb.com/react/react.usecontract

16. useContractRead [Електронний ресурс] - Режим доступу до ресурсу: https://portal.thirdweb.com/react/react.usecontractread

17. useContractWrite [Електронний ресурс] - Режим доступу до ресурсу: https://portal.thirdweb.com/react/react.usecontractwrite

18. React createContext [Електронний ресурс] - Режим доступу до ресурсу: https://react.dev/reference/react/createContext

19. React useContext [Електронний ресурс] - Режим доступу до ресурсу: https://react.dev/reference/react/useContext

20. React Context Hooks [Електронний ресурс] - Режим доступу до ресурсу: https://react.dev/reference/react#context-hooks

21. WEI ethers [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ethers.org/v5/api/utils/bignumber/#BigNumber--notes-safenumbers

22. ethers.js documentation [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ethers.org/v5/

23. ethers.js formatEther [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ethers.org/v5/api/utils/display-logic/#utils-formatEther

24. MetaMask [Електронний ресурс] – Режим доступу до ресурсу: https://metamask.io/

25. GTMetrix [Електронний ресурс] – Режим доступу до ресурсу: https://gtmetrix.com/

26. Pagespeed [Електронний ресурс] – Режим доступу до ресурсу: https://pagespeed.web.dev/

ДОДАТКИ

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

contract CrowdFunding {
    struct Campaign {
        uint256 id;
        address owner;
        string title;
        string description;
        uint256 target;
        uint256 deadline;
        uint256 amountCollected;
        string image;
        uint256 category;
        address[] donators;
        uint256[] donations;
        uint256[] commissions;
    }

    mapping(uint256 => Campaign) public campaigns;

    uint256 public numberOfCampaigns = 0;

    function createCampaign(
        address _owner,
        string memory _title,
        string memory _description,
        uint256 _target,
        uint256 _deadline,
        uint256 _category,
        string memory _image
    ) public returns (uint256) {
        uint256 newId = numberOfCampaigns;
        Campaign storage campaign = campaigns[newId];

        require(
            _deadline / 1000 > block.timestamp,
            "The deadline should be a date in the future."
        );
        campaign.id = newId;
        campaign.owner = _owner;
        campaign.title = _title;
        campaign.description = _description;
        campaign.target = _target;
        campaign.deadline = _deadline;
        campaign.amountCollected = 0;
        campaign.image = _image;
        campaign.category = _category;

        numberOfCampaigns++;

        return newId;
    }
```

```
    function donateToCampaign(
        uint256 _id,
        address _commissionAddress
    ) public payable {
        uint256 amount = msg.value;

        Campaign storage campaign = campaigns[_id];

        require(
            campaign.deadline > block.timestamp,
            "The deadline for this campaign has already passed."
        );

        uint256 commission = (amount * 10) / 100; // Calculate 10% commission
        uint256 donationAmount = amount - commission;

        campaign.donators.push(msg.sender);
        campaign.donations.push(donationAmount);
        campaign.commissions.push(commission);

        (bool sent, ) = payable(campaign.owner).call{value: donationAmount}("");

        require(sent, "Failed to send Ether to campaign owner.");

        (bool commissionSent, ) = payable(_commissionAddress).call{
            value: commission
        }("");

        require(commissionSent, "Failed to send commission.");

        campaign.amountCollected += donationAmount;
    }

    function getDonators(
        uint256 _id
    ) public view returns (address[] memory, uint256[] memory) {
        Campaign storage campaign = campaigns[_id];
        uint256 numDonators = campaign.donators.length;

        address[] memory donators = new address[](numDonators);
        uint256[] memory donations = new uint256[](numDonators);

        for (uint256 i = 0; i < numDonators; i++) {
            donators[i] = campaign.donators[i];
            donations[i] = campaign.donations[i];
        }

        return (donators, donations);
    }

    function getAllDonationsForCampaigns()
        public
        view
        returns (
```

```
            Campaign[] memory,
            address[] memory,
            address[][] memory,
            uint256[][] memory,
            uint256[][] memory
        )
    {
        Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);
        address[] memory authors = new address[](numberOfCampaigns);
        address[][] memory allDonators = new address[][](numberOfCampaigns);
        uint256[][] memory allDonations = new uint256[][](numberOfCampaigns);
        uint256[][] memory allCommissions = new uint256[][](numberOfCampaigns);

        for (uint256 i = 0; i < numberOfCampaigns; i++) {
            Campaign storage campaign = campaigns[i];
            uint256 numDonators = campaign.donators.length;

            address[] memory donators = new address[](numDonators);
            uint256[] memory donations = new uint256[](numDonators);
            uint256[] memory commissions = new uint256[](numDonators);

            for (uint256 j = 0; j < numDonators; j++) {
                donators[j] = campaign.donators[j];
                donations[j] = campaign.donations[j];
                commissions[j] = campaign.commissions[j];
            }

            allCampaigns[i] = campaign;
            authors[i] = campaign.owner;
            allDonators[i] = donators;
            allDonations[i] = donations;
            allCommissions[i] = commissions;
        }

        return (
            allCampaigns,
            authors,
            allDonators,
            allDonations,
            allCommissions
        );
    }

    function getCampaigns() public view returns (Campaign[] memory) {
        Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);

        for (uint i = 0; i < numberOfCampaigns; i++) {
            Campaign storage item = campaigns[i];

            allCampaigns[i] = item;
        }

        return allCampaigns;
    }
```

```
    function getCampaignById(
        uint256 _id
    ) public view returns (Campaign memory) {
        require(_id < numberOfCampaigns, "Invalid campaign ID.");

        return campaigns[_id];
    }
}
```

```
import React, { useContext, createContext, ReactNode } from "react";

import {
  useAddress,
  useContract,
  useMetamask,
  useContractWrite,
} from "@thirdweb-dev/react";
import { ethers } from "ethers";
import { FeesWeb3 } from "@/types/fees.type";

interface CreateCampaign {
  title: string;
  description: string;
  target: ethers.BigNumber;
  deadline: number;
  image: string;
  category: number;
}

interface Donation {
  donator: string;
  donation: string;
}

interface ContractContextProps {
  address: string | undefined;
  contract: any;
  connect: any;
  createCampaign: (form: CreateCampaign) => Promise<unknown>;
  getCampaigns: () => Promise<FeesWeb3[]>;
  getCampaign: (pId: number) => Promise<FeesWeb3 | undefined>;
  getUserCampaigns: () => Promise<FeesWeb3[]>;
  donate: (pId: any, amount: any) => Promise<any>;
  getDonations: (pId: any) => Promise<Donation[]>;
  getAllDonationsForCampaigns: () => any;
}

interface ContractContextProviderProps {
  children: ReactNode;
}

const ContractContext = createContext<ContractContextProps>(
```

```
  {} as ContractContextProps
);

export const ContractContextProvider = ({
  children,
}: ContractContextProviderProps) => {
  const { contract } = useContract(
    "0x7864907848324cfd063A8aE71023E246dccBBbD4"
  );
  const { mutateAsync: createCampaign } = useContractWrite(
    contract,
    "createCampaign"
  );

  const address = useAddress();
  const connect = useMetamask();

  const publishCampaign = async (form: CreateCampaign) => {
    try {
      const data = await createCampaign({
        args: [
          address, // owner
          form.title, // title
          form.description, // description
          form.target,
          form.deadline, // deadline,
          form.category,
          form.image,
        ],
      });

      return data;
    } catch (error) {
      console.log("contract call failure", error);
    }
  };

  const getCampaigns = async () => {
    const campaigns = await contract?.call("getCampaigns");

    const parsedCampaings = campaigns?.map((campaign: any, i: any) => ({
      pId: campaign.id.toNumber(),
      owner: campaign.owner,
      title: campaign.title,
      description: campaign.description,
      //@ts-ignore
      goal: ethers.utils.formatEther(
        campaign.target.sub(campaign.target.mod(1e14))
      ),
      deadline: campaign.deadline.toNumber(),
      //@ts-ignore
      raised: ethers.utils.formatEther(
        campaign.amountCollected.sub(campaign.amountCollected.mod(1e14))
      ),
```

```
      image: campaign.image,
      category: campaign.category.toNumber(),
    }));

    return parsedCampaings?.filter(
      (item: any) => item.deadline > new Date().getTime() / 1000
    );
  };

  const getCampaign = async (pId: number) => {
    const campaign = await contract?.call("getCampaignById", [pId]);

    if (campaign) {
      const parsedCampaings = {
        pId: campaign.id,
        owner: campaign.owner,
        title: campaign.title,
        description: campaign.description,
        //@ts-ignore
        goal: ethers.utils.formatEther(
          campaign.target.sub(campaign.target.mod(1e14))
        ),
        deadline: campaign.deadline.toNumber(),
        //@ts-ignore
        raised: ethers.utils.formatEther(
          campaign.amountCollected.sub(campaign.amountCollected.mod(1e14))
        ),
        image: campaign.image,
        category: campaign.category,
      };
      return parsedCampaings;
    }
  };

  const getUserCampaigns = async () => {
    const allCampaigns = await getCampaigns();

    const filteredCampaigns = allCampaigns.filter(
      (campaign: any) => campaign.owner === address
    );

    return filteredCampaigns;
  };

  const donate = async (pId: any, amount: any) => {
    try {
      const data = await contract?.call(
        "donateToCampaign",
        [pId, "0x7a254602c85235E54bf85587eb13Cdb09238DA51"],
        {
          //@ts-ignore
          value: ethers.utils.parseEther(String(parseFloat(amount))),
        }
      );
```

```
    return data;
  } catch {
    return;
  }
};

const getDonations = async (pId: any) => {
  const donations = await contract?.call("getDonators", pId);
  const numberOfDonations = donations[0].length;

  const parsedDonations = [];

  for (let i = 0; i < numberOfDonations; i++) {
    parsedDonations.push({
      donator: donations[0][i],
      //@ts-ignore
      donation: ethers.utils.formatEther(donations[1][i].toString()),
    });
  }

  return parsedDonations;
};

const getAllDonationsForCampaigns = async () => {
  const data = await contract?.call("getAllDonationsForCampaigns");

  const parsedData = [];
  if (data) {
    for (let i = 0; i < data[0].length; i++) {
      const campaign = data[0][i];
      const author = data[1][i];
      const campaignDonators = data[2][i];
      const campaignDonations = data[3][i];
      const campaignCommissions = data[4][i];

      const parsedCampaign: any = {
        id: campaign.id,
        title: campaign.title,
        owner: author,
        goal: campaign.target,
        deadline: campaign.deadline.toNumber(),
        raised: ethers.utils.formatEther(
          campaign.amountCollected.sub(campaign.amountCollected.mod(1e14))
        ),
        category: campaign.category.toNumber(),
        donations: [],
      };

      for (let j = 0; j < campaignDonators.length; j++) {
        const donator = campaignDonators[j];
        const donation = ethers.utils.formatEther(
          campaignDonations[j].toString()
        );
        const commission = ethers.utils.formatEther(
```

```
            campaignCommissions[j].toString()
          );

          parsedCampaign.donations.push({ donator, donation, commission });
        }

        parsedData.push(parsedCampaign);
      }
    }
    return parsedData;
  };

  return (
    <ContractContext.Provider
      value={{
        address,
        contract,
        connect,
        createCampaign: publishCampaign,
        getCampaigns,
        getCampaign,
        getUserCampaigns,
        donate,
        getDonations,
        getAllDonationsForCampaigns,
      }}
    >
      {children}
    </ContractContext.Provider>
  );
};

export const useContractContext = () => useContext(ContractContext);

import { FeesWeb3 } from "@/types/fees.type";
import { useState, useEffect } from "react";

const useDataFilter = (
  data: FeesWeb3[],
  initialQuery = "",
  initialSortBy = "",
  initialFilterCategory = "all"
) => {
  const [query, setQuery] = useState(initialQuery);
  const [sortBy, setSortBy] = useState(initialSortBy);
  const [filterCategory, setFilterCategory] = useState<number | string>(
    initialFilterCategory
  );
  const [results, setResults] = useState<Array<FeesWeb3>>([]);

  useEffect(() => {
    let filteredData = data.filter(
      (item) =>
        item.title.toLowerCase().includes(query.toLowerCase()) ||
```

```
        item.description.toLowerCase().includes(query.toLowerCase())
    );

    if (filterCategory) {
      if (filterCategory === "all" && typeof filterCategory === "string") {
        filteredData = filteredData;
      } else {
        filteredData = filteredData.filter(
          (item) => item.category === filterCategory
        );
      }
    }

    switch (sortBy) {
      case "name-asc":
        filteredData = filteredData.sort((a, b) =>
          a.title.localeCompare(b.title)
        );
        break;
      case "name-desc":
        filteredData = filteredData.sort((a, b) =>
          b.title.localeCompare(a.title)
        );
        break;
      case "id-asc":
        filteredData = filteredData.sort((a, b) => a.pId - b.pId);
        break;
      case "id-desc":
        filteredData = filteredData.sort((a, b) => b.pId - a.pId);
        break;
      default:
        break;
    }

    setResults(filteredData);
  }, [data, query, sortBy, filterCategory]);

  const handleSearch = (newQuery: string) => {
    setQuery(newQuery);
  };

  const handleSortBy = (value: string) => {
    setSortBy(value);
  };

  const handleFilterByCategory = (value: number | string) => {
    setFilterCategory(value);
  };

  return {
    query,
    results,
    handleSearch,
    sortBy,
```

```
    handleSortBy,
    filterCategory,
    handleFilterByCategory,
  };
};

export default useDataFilter;

import { useContractContext } from "@/context/ContractContext";
import { useGetCategoriesQuery } from "@/store/reducers/category/categoryApi";
import {
  Box,
  Button,
  Card,
  CardContent,
  FormControl,
  InputLabel,
  MenuItem,
  Select,
  Typography,
} from "@mui/material";
import { useSession } from "next-auth/react";
import { useEffect, useState } from "react";
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from "chart.js";
import { Pie } from "react-chartjs-2";
import { setDatasets } from "react-chartjs-2/dist/utils";
ChartJS.register(ArcElement, Tooltip, Legend);

const Statistics: React.FC = () => {
  const session = useSession();
  const { getAllDonationsForCampaigns, address, connect } =
    useContractContext();
  const [allDonations, setAllDonations] = useState<Array<any>>([]);
  const [isLoadingDonations, setIsLoadingDonations] = useState(false);
  const [totalProfit, setTotalProfit] = useState(0);
  const [ammountCollected, setAmmountCollected] = useState(0);
  const [totalDonates, setTotalDonates] = useState(0);
  const [selectedCategory, setSelectedCategory] = useState<number | string>();
  const { data: categories } = useGetCategoriesQuery();
  const [labels, setLabels] = useState<string[]>([]);

  const commissionByCategory: any = {};
  const commissionByFees: any = {};

  useEffect(() => {
    const getDonations = async () => {
      setIsLoadingDonations(true);
      const res = await getAllDonationsForCampaigns();
      if (res) {
        setAllDonations(res);
        setIsLoadingDonations(false);
      }
    };
    getDonations();
```

```
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [getAllDonationsForCampaigns]);

  useEffect(() => {
    for (var member in commissionByCategory)
      delete commissionByCategory[member];
    for (var member in commissionByFees) delete commissionByFees[member];
    setTotalDonates(0);
    setAmmountCollected(0);
    setTotalDonates(0);
    if (allDonations) {
      if (session.data?.user.role === "Admin") {
        allDonations.map((item) => {
          item.donations.map((donation: any) => {
            setTotalProfit((prev) => (prev += Number(donation?.commission)));
            setAmmountCollected((prev) => (prev += Number(donation?.donation)));
          });
          setTotalDonates((prev) => (prev += item.donations.length));
        });
      } else {
        allDonations
          .filter((item) => item.owner === address)
          .map((item) => {
            item.donations.map((donation: any) => {
              setTotalProfit((prev) => (prev += Number(donation?.commission)));
              setAmmountCollected(
                (prev) => (prev += Number(donation?.donation))
              );
            });
            setTotalDonates((prev) => (prev += item.donations.length));
          });
      }
    }

    //eslint-disable-next-line
  }, [allDonations]);

  session.data?.user.role === "Admin"
    ? allDonations.forEach((item) => {
        const Title = item.title;
        const donations = item.donations;

        let commissionSum = 0;
        donations.forEach((donation: any) => {
          commissionSum += parseFloat(donation.commission);
        });

        if (commissionByFees[" " + Title]) {
          commissionByFees[" " + Title] += commissionSum;
        } else {
          commissionByFees[" " + Title] = commissionSum;
        }
      })
    : allDonations
```

```
      .filter((item) => item.owner === address)
      .forEach((item) => {
        const Title = item.title;
        const donations = item.donations;

        let commissionSum = 0;
        donations.forEach((donation: any) => {
          commissionSum += parseFloat(donation.donation);
        });

        if (commissionByFees[" " + Title]) {
          commissionByFees[" " + Title] += commissionSum;
        } else {
          commissionByFees[" " + Title] = commissionSum;
        }
      });

session.data?.user.role === "Admin"
  ? allDonations.forEach((item) => {
      const category = item.category;
      const donations = item.donations;

      let commissionSum = 0;
      donations.forEach((donation: any) => {
        commissionSum += parseFloat(donation.commission);
      });

      if (commissionByCategory[" " + category]) {
        commissionByCategory[" " + category] += commissionSum;
      } else {
        commissionByCategory[" " + category] = commissionSum;
      }
    })
  : allDonations
      .filter((item) => item.owner === address)
      .forEach((item) => {
        const category = item.category;
        const donations = item.donations;

        let commissionSum = 0;
        donations.forEach((donation: any) => {
          commissionSum += parseFloat(donation.donation);
        });

        if (commissionByCategory[" " + category]) {
          commissionByCategory[" " + category] += commissionSum;
        } else {
          commissionByCategory[" " + category] = commissionSum;
        }
      });

useEffect(() => {
  setLabels([]);
  if (categories) {
```

```
    categories.map((item) => {
      setLabels((prev) => [...prev, item.title]);
    });
  }
}, [categories]);

const getColors = (labels: Array<any>) => {
  const bgColors: any = [];
  const borderColors: any = [];
  for (let i = 0; i < labels.length; i++) {
    const r = Math.floor(Math.random() * 255);
    const g = Math.floor(Math.random() * 255);
    const b = Math.floor(Math.random() * 255);
    bgColors.push("rgba(" + r + ", " + g + ", " + b + ", 0.5");
    borderColors.push("rgba(" + r + ", " + g + ", " + b + ", 1");
  }
  return {
    backgroundColor: bgColors,
    borderColor: borderColors,
  };
};

const dataCategory = {
  labels: [...labels],
  datasets: [
    {
      label: "total colected",
      data: [...Object.values(commissionByCategory)],
      ...getColors(labels),
      Width: 1,
    },
  ],
};

const dataFees = {
  labels: [...allDonations.map((item) => item.title)],
  datasets: [
    {
      label: "total colected",
      data: [...Object.values(commissionByFees)],
      ...getColors(allDonations.map((item) => item.title)),
      Width: 1,
    },
  ],
};

return (
  <>
    {session.data?.user.role === "Admin" ||
      (session.data?.user.role === "User" && address) ? (
        <>
          {(session.data?.user.role !== "User" && allDonations.length > 0) ||
          (session.data?.user.role === "User" &&
            allDonations.filter((item) => item.owner === address).length >
```

```
          0) ? (
        <Box sx={{ p: "20px", width: "calc(100% - 260px)" }}>
          <div
            style={{
              width: "100%",
              display: "flex",
              justifyContent: "space-between",
            }}
          >
            <Card>
              <CardContent>
                <Typography
                  variant="h6"
                  color="textSecondary"
                  gutterBottom
                >
                  Total Feeses
                </Typography>
                <Typography variant="h4">
                  {session.data.user.role === "User"
                    ? allDonations.filter(
                        (item) => item.owner === address
                      ).length
                    : allDonations.length}
                </Typography>
              </CardContent>
            </Card>

            {session && session.data?.user.role !== "User" && (
              <Card>
                <CardContent>
                  <Typography
                    variant="h6"
                    color="textSecondary"
                    gutterBottom
                  >
                    Total Commission Collected
                  </Typography>
                  <Typography variant="h4">{totalProfit.toFixed(5)}</Typography>
                </CardContent>
              </Card>
            )}

            <Card>
              <CardContent>
                <Typography
                  variant="h6"
                  color="textSecondary"
                  gutterBottom
                >
                  Total Donates
                </Typography>
                <Typography variant="h4">{totalDonates}</Typography>
              </CardContent>
```

```
              </Card>

              <Card>
                <CardContent>
                  <Typography
                    variant="h6"
                    color="textSecondary"
                    gutterBottom
                  >
                    Total Amount Collected
                  </Typography>
                  <Typography variant="h4">{ammountCollected.toFixed(5)}</Typography>
                </CardContent>
              </Card>
            </div>
            <Box
              sx={{
                margin: "40px auto",
                display: "flex",
                justifyContent: "space-between",
              }}
            >
              <Box sx={{ width: "40%" }}>
                <Pie data={dataCategory} width={50} height={50} />
              </Box>
              <Box sx={{ width: "40%" }}>
                {" "}
                <Pie data={dataFees} width={50} height={50} />
              </Box>
            </Box>
          </Box>
        ) : (
          <Box sx={{ position: "relative", width: "calc(100% - 260px)" }}>
            <Typography
              variant="h6"
              component="h3"
              gutterBottom
              sx={{
                position: "absolute",
                top: "50%",
                left: "50%",
                transform: "translate(-50%, -50%)",
              }}
            >
              No statistic now here
            </Typography>
          </Box>
        )}
      </>
    ) : (
      <>
        <Box sx={{ position: "relative", width: "calc(100% - 260px)" }}>
          <Button
            variant="contained"
```

```
                sx={{
                  bgcolor: "#02a95c",
                  color: "white",
                  position: "absolute",
                  top: "50%",
                  left: "50%",
                  transform: "translate(-50%, -50%)",
                }}
                onClick={connect}
              >
                Connect Wallet
              </Button>
            </Box>
          </>
        )}
      {/* {session.data?.user.role === "User" && <>user</>} */}
    </>
  );
};

export default Statistics;
```