КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

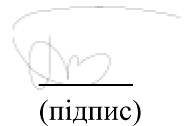
за спеціальністю 122 Комп’ютерні науки

на тему:

ПОБУДОВА СМАРТ-КОНТРАКТУ ТА ІНТЕРФЕЙСУ ДЛЯ ВЕБ-ЗАСТОСУНКУ МОНІТОРИНГУ БАНКІВСЬКИХ РАХУНКІВ

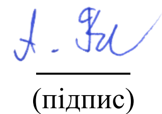
(BUILDING A SMART CONTRACT AND INTERFACE FOR THE BANK ACCOUNT MONITORING WEB APPLICATION)

Виконав студент 4-го курсу групи МІ-4
Тірік Олександр



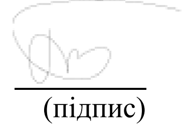
(підпис)

Науковий керівник:
Асистент
Олексій Федорус



(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент 

(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 38 сторінок, 20 рисунків, 15 джерел посилання.

Назва роботи: Побудова смарт-контракту та інтерфейсу для веб-застосунку моніторингу банківських рахунків.

Мета роботи: розробити сервіс, який візьме на себе відповідальність за публічну демонстрацію стану банківських рахунків, а також надійне зберігання всієї актуальної інформації в блокчейні.

План роботи: дослідити існуючі блокчейни та обрати потрібний для цього проекту, дослідити як правильно розробити та протестувати смарт контракти на обраному блокчейні. Реалізувати графічний інтерфейс для проекту.

Результат роботи: робочий графічний інтерфейс для сервісу відображення стану банківських рахунків. Також робочий смарт контракт, який зберігає актуальну інформацію про транзакції банківського акаунту.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5
ВСТУП 6
РОЗДІЛ 1. ВИБІР БЛОКЧЕЙНУ ДЛЯ РОЗРОБКИ 8
1.1 Біткоїн 8
1.2 Ефіріум 9
1.3 Солана 9
1.4 Виновок 10
РОЗДІЛ 2. РОЗРОБКА БЛОКЧЕЙН ПРОГРАМИ У МЕРЕЖІ СОЛАНА 11
2.1 Огляд моделі програмування 11
2.2 Огляд способів розробки програми 13
2.3 Розробка програми 14
2.3.1 Метод ініціалізації 15
2.3.2 Метод додавання транзакції 17
2.4 Тестування програми 19
2.4.1 Тестування методу ініціалізації 19
2.4.2 Тестування методу додавання транзакції 20
РОЗДІЛ 3. РОЗРОБКА ГРАФІЧНОГО ВЕБ ІНТЕРФЕЙСУ 22
3.1 Огляд фреймворку для розробки 22
3.2 Створення дизайну у сервісі Figma 23
3.3 Розробка компоненів інтерфейсу 24
3.3.1 Компонент інформації про проект 24
3.3.2 Компонент графік 25
3.3.3 Компонент реквізит 26
3.3.4 Компонент транзакції 27
3.4 Структури маніпулювання даними 29
3.4.1 Об’єкти станів 30
3.4.2 Об’єкти роботи з запитами 31

3.4.3 Веб-сокет хендлери 32
3.5 Контейнер головної сторінки 33
ВИСНОВКИ 35
ДОДАТКИ 36
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 37

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

URL - Uniform Resource Locator

API - Application Programming Interface

ВСТУП

Оцінка сучасного стану об’єкта розробки. Протягом усієї історії існувало багато ідей, на реалізацію яких потрібно було б багато грошей. Саме тому люди стали кооперуватися та збирати гроші для реалізації цих ідей. З розвитком економіки методи збирання фінансів змінювалися. Зараз 2022 рік, у нас є безліч фінансових інструментів, завдяки яким можна зручно відправити гроші куди завгодно, навіть не бравши їх в руки. Останнім часом питання збору грошей на щось дуже затребуване, люди збирають гроші на лікування, на відновлення будівель, на підтримку армії. Всі цивілізовані держави світу вже придумали як регулювати це питання, зазвичай створюються юридичні особи, які є збирачами грошей, а потім вони витрачають їх на цілі, які вони поставили. Найчастіше такі юридичні особи не оподатковуються, і за статистикою вони найчастіше задіяні у корупційних схемах. Юридичні особи це лише один із варіантів збору грошей, гроші так само збирають і фізичні особи. Зараз, коли держава не в силі виконувати всі свої обов'язки, люди не вдаються до створення юридичних осіб та збирають гроші на свої банківські картки. Якщо дивитися абстрактно на цю ситуацію, тоді можна побачити, що гроші збирають не лише люди в інтернеті, але також і на вулиці, на касах у магазині та в інших місцях.

Напевно кожна людина коли-небудь жертвувала гроші на благодійність наприклад, а чи запитували ви: а скільки грошей вже зібрано? Зазвичай на це запитання ви або не дізнаєтесь відповіді, або дізнаєтеся її від того, хто збирає ці гроші. Так само є ще варіант проходу через кола пекла бюрократії, і дізнатися про стан рахунків певної юридичної особи, який займається благодійністю, у держави, але як показує практика - це неможливо, а у фізичних осіб такого варіанту і зовсім немає.

Актуальність роботи та підстави для її виконання. Кількість людей і організацій які зараз збирають гроші на щось зашкалює. Зайдіть в будь-який банківський додаток і перегляньте розділ благодійність, кожен може побачити, що список буде великий. Зайдіть на сайт або сторінку в соціальних мережах

будь-якого притулку, лікарні, військового, волонтера, і навіть університету, ви побачите, що зараз терміново потрібні гроші на: корм для тварин, медикаменти, прилади нічного зору, їжу тощо. Немає жодного чесного та зручного способу довести скільки грошей вже було зібрано на мету. А також, у разі шахрайства з боку людини або організації, яка збирає гроші, немає зручного шляху доказу, що людина справді відправляла гроші на рахунок.

Мета й завдання роботи. Метою кваліфікаційної роботи є розробка веб-інтерфейсу та програми блокчейну, що буде зберігати актуальні дані про стан банківського рахунку. Задля успішного виконання роботи, треба:

- Дослідити різні надійні блокчейни задля подальшої роботи з ними
- Розробити програму для децентралізованого зберігання інформації
- Розробити графічний веб-інтерфейс
- Проаналізувати результат роботи

Об’єкт, методи й засоби розробки. Об’єктом розробки програмного забезпечення є веб-інтерфейс та блокчейн смарт-контракт.

Можливі сфери застосування. Розроблений продукт можна використовувати де завгодно, де метою завдання є загальний збір грошей. Найактуальніша на даний момент сфера використання даного продукту – це благодійність

РОЗДІЛ 1. ВИБІР БЛОКЧЕЙНУ ДЛЯ РОЗРОБКИ

Блокчейн[1][2] - це список записів, званих блоками, які надійно пов'язані криптографією. Кожен блок містить криптографічний хеш попереднього блоку, мітку часу та транзакцію. Тимчасова мітка доводить, що дані транзакцій є, коли вони публікуються для введення хеша блоку. Оскільки кожен блок містить інформацію про попередній блок, кожен додатковий блок створює ланцюжок, що посилює попередні події. Таким чином, блокчейни стійкі до зміни своїх даних, оскільки після запису дані в будь-якому блоці не можуть бути змінені заднім числом без зміни всіх наступних блоків[3].

Проаналізуємо 3 різних блокчейни і виберемо один із них для подальшої розробки. Вибирати слід надійні блокчейни, я вважаю, що надійність визначається капіталізацією, і кількістю часу, що існує блокчейн. Для аналізу я вибрав такі блокчейни: Bitcoin, Ethereum, Solana.

1.1 Біткоїн

Біткоїн[4][5] - це децентралізована цифрова валюта, яку можна переводити через однорангову мережу Біткоїн. Криптовалюта була винайдена у 2008 році невідомою особою чи групою осіб під ім'ям Сатоші Накамото. Валюта була запущена в 2009 році, коли її було випущено як програмне забезпечення з відкритим вихідним кодом. Біткоїни створюються як винагорода за процес майнінгу. Їх можна обміняти на інші валюти, товари та послуги. Біткоїн критикувався за його використання в незаконних транзакціях, велику кількість електроенергії, що використовується для майнінгу (і, отже, вуглецевий слід), волатильність цін та крадіжка на біржі. Деякі інвестори та економісти у різний час описували його як спекулятивний міхур. Біткойн офіційно використовується в тій чи іншій якості кількома місцевими органами влади та урядами штатів, а дві країни, Сальвадор і Центральноафриканська Республіка, прийняли його як законний платіжний засіб.

Зараз Біткоїн має найбільшу капіталізацію у більш ніж 500 мілліардів доларів. Ця криптовалюта зарекомендувала себе як безпечну, але є одна головна проблема, це відсутність реалізації смарт контрактів. Відповідно, ми не будемо в подальшому розглядати цю криптовалюту для використання в цій роботі.

1.2 Ефіріум

Ethereum[6] – це децентралізований блокчейн з відкритим вихідним кодом та можливостями смарт-контракт

ів. Ефір – це власна криптовалюта платформи. Серед криптовалют ринкова капіталізація ефіру поступається лише біткоїну. Ethereum був задуманий у 2013 році програмістом Віталіком Бутеріним. Іншими засновниками Ethereum є Гевін Вуд, Чарльз Хоскінсон, Ентоні Ді Іоріо та Джозеф Любін. Розробка почалася у 2014 році, вона була зібрана за рахунок краудфандингу та запущена 30 липня 2015 року. Ethereum дозволяє будь-кому розгортати на ньому постійні та незмінні децентралізовані програми, з якими користувачі можуть взаємодіяти. Додатки децентралізованого фінансування (DeFi) надають широкий спектр фінансових послуг без потреби у типових фінансових посередниках, таких як брокерські контори, біржі або банки, дозволяючи, наприклад, користувачам криптовалюти брати кредити під заставу своїх активів або під проценти. Ethereum також дозволяє користувачам створювати та обмінювати NFT, унікальні токени, які мають право власності на будь-яку кількість інституційно визнаних пов'язаних активів або привілеїв. Крім того, багато інших криптовалют використовують стандарт токенів ERC-20 поверх блокчейна Ethereum і використовують платформу для початкових пропозицій монет[7].

Ефіріум дуже популярний але для додавання інформації у блокчейн потрібно платити за те щоб транзакція пройшла, зараз коміссії у блокчейні Ethereum дуже високі (більше 5 умовних одиниць), саме за це ми не можемо обрати цей варіант.

1.3 Солана

Solana[8] - це проект з відкритим вихідним кодом, що реалізує новий

високопродуктивний блокчейн. Фонд Солана, що базується в Женеві, Швейцарія, підтримує проект із відкритим вихідним кодом.

Централізована база даних може обробляти 710 000 транзакцій за секунду у стандартній гігабітній мережі, якщо середній розмір транзакцій не перевищує 176 байт. Використовуючи техніку розподілених систем, яка називається оптимістичним контролем паралелізму [16], централізована база даних може також реплікувати себе і підтримувати високу доступність без значного уповільнення транзакцій. Солана продемонструвала, що ті ж теоретичні обмеження застосовуються до блокчейнів у змагальних мережах. Ключовий інгредієнт? Знайти спосіб розділити момент, коли вузли не можуть залежати один від одного. Правильно, і відповідь на це, вузли повинні покладатися на час.

Масштабованість Solana гарантує, що транзакції залишаються менше ніж 0,01 дол. США як для розробників, так і для користувачів.

1.4 Виновок

Проаналізувавши три різні блокчейни, ми можемо зробити висновок, що солана є для нас найоб'єктивнішим вибором. Біткоїн ми не можемо вибрати, тому що він не представляє функціоналу програм(смарт контрактів), за допомогою яких ми могли б зберігати дані в блокчейні. Ефіріум, у свою чергу, надає смарт контракти на відміну від біткоїну, але ціни на транзакції в мережі ефіру дуже високі, що ми не можемо дозволити для реалізації цієї роботи. Солана в свою чергу є ідеальним варіантом для цього завдання, пропускна спроможність мережі солани є дуже високою, ціни на транзакції низькі, смарт контракти підтримує. Можемо сказати, що розробка блокчейн частини цього проекту буде на блокчейні Солана.

РОЗДІЛ 2. РОЗРОБКА БЛОКЧЕЙН ПРОГРАМИ У МЕРЕЖІ СОЛАНА

2.1 Огляд моделі програмування

Програма взаємодіє з кластером Solana, надсилаючи йому транзакції з однією або кількома інструкціями. Середовище виконання Solana передає ці інструкції програмам, розгорнутим розробниками програм заздалегідь. Інструкція може, наприклад, вказати програмі передати криптовалюту з одного облікового запису в інший або створити інтерактивний контракт, який регулює спосіб передачі криптовалюти. Інструкції виконуються послідовно й атомарно для кожної транзакції. Якщо будь-яка інструкція недійсна, усі зміни облікового запису в транзакції відхиляються.

Виконання програми починається з подання транзакції в кластер. Середовище виконання Solana виконає програму для обробки кожної з інструкцій, що містяться в транзакції, по порядку й атомарно.

Транзакція містить компактний масив підписів, за яким слідує повідомлення. Кожен елемент у масиві підписів є цифровим підписом даного повідомлення. Середовище виконання Solana перевіряє, що кількість підписів відповідає числу в перших 8 бітах заголовка повідомлення. Він також перевіряє, що кожен підпис був підписаний закритим ключем, що відповідає відкритому ключу з тим самим індексом у масиві адрес облікового запису повідомлення.

Кожен цифровий підпис має двійковий формат ed25519 і споживає 64 байти.

Повідомлення містить заголовок, за яким слід компактний масив адрес облікових записів, за яким слідує нещодавній блок-хеш, за яким слід компактний масив інструкцій.

Кожна інструкція визначає одну програму, підмножину рахунків транзакції, які мають бути передані програмі, і масив байтів даних, який передається програмі. Програма інтерпретує масив даних і оперує рахунками, зазначеними в інструкції. Програма може повернутися успішно або з кодом помилки. Повернення помилки призводить до того, що вся транзакція негайно зазнає невдачі.

Якщо програмі потрібно зберігати стан між транзакціями, вона робить це за допомогою акаунтів. Акаунти подібні до файлів в операційних системах, таких як Linux, оскільки вони можуть містити довільні дані, які зберігаються протягом усього терміну життя програми. Так само, як і файл, акаунт містить метадані, які повідомляють середовищі виконання, кому і як дозволено доступ до даних.

На відміну від файлу, акаунт містить метадані на весь час існування файлу. Цей час життя виражається кількома дробовими нативними лексемами, які називаються лампортами. Рахунки зберігаються в пам'яті валідатора і платять «оренду», щоб залишитися там. Кожен валідатор періодично сканує всі рахунки та збирає орендну плату. Будь-який обліковий запис, який знижується до нуля лампортів, очищається. Рахунки також можуть бути позначені як звільнені від оренди, якщо вони містять достатню кількість лампортів.

Так само, як користувач Linux використовує шлях для пошуку файлу, клієнт Solana використовує адресу для пошуку акаунту. Адреса є 256-бітним відкритим ключем.

Середовище виконання дозволяє програмі власника лише дебетувати рахунок або змінювати його дані. Потім програма визначає додаткові правила щодо того, чи може клієнт змінювати облікові записи, якими він володіє. У випадку з системною програмою, вона дозволяє користувачам передавати сигнали, розпізнаючи підписи транзакцій. Якщо він бачить, що клієнт підписав транзакцію за допомогою закритого ключа пари ключів, він знає, що клієнт авторизував передачу токена.

Іншими словами, весь набір облікових записів, що належать даній програмі, можна розглядати як сховище ключ-значення, де ключ — це адреса облікового запису, а значення — це довільні двійкові дані програми. Автор програми може вирішити, як керувати всім станом програми, можливо, кількома акаунтами.

Після того, як середовище виконання виконує кожну з інструкцій транзакції, воно використовує метадані акаунту, щоб перевірити, чи не було порушено політику доступу. Якщо програма порушує політику, середовище виконання відкидає всі зміни акаунту, внесені всіма інструкціями в транзакції, і позначає

транзакцію як невдалу[9].

2.2 Огляд способів розробки програми

Розробники можуть писати та розгортати власні програми на блокчейні Solana.

Програми Solana компілюються через інфраструктуру компілятора LLVM у формат виконуваного файлу з можливістю зв'язування (ELF), що містить варіацію байт-коду Berkeley Packet Filter (BPF).

Оскільки Solana використовує інфраструктуру компілятора LLVM, програма може бути написана будь-якою мовою програмування, яка може орієнтуватися на бекенд BPF LLVM. Наразі Solana підтримує написання програм на Rust та C/C++.

BPF надає ефективний набір інструкцій, який може виконуватися у інтерпретованій віртуальній машині або як ефективні скомпільовані вчасно скомпільовані інструкції.

BPF використовує стекові фрейми замість змінного покажчика стека. Кожен кадр стека має розмір 4 КБ.

Програми мають доступ до кучі під час виконання безпосередньо на C або через API розподілу Rust. Щоб полегшити швидке розподілення, використовується простий 32 КБ бумпінг. Куча не підтримує free або realoc, тому використовувати її потрібно з розумом.

Внутрішньо програми мають доступ до області пам'яті розміром 32 КБ, починаючи з віртуальної адреси 0x300000000, і можуть реалізувати власну купу на основі конкретних потреб програми[10].

Спільні об'єкти програми не підтримують спільні дані для запису. Програми розподіляються між кількома паралельними виконаннями з використанням одного спільного коду та даних лише для читання. Це означає, що розробники не повинні включати в програми будь-які статичні записувані або глобальні змінні.

На високому рівні пам'ять всередині кластера Solana можна розглядати як монолітну купу даних. Смарт-контракти на Solana («програми» на жаргоні Solana)

мають доступ до власної частини цієї купи.

Хоча програма може читати будь-яку частину глобальної купи, якщо програма намагається записати в частину купи, яка не є її, середовище виконання Solana призводить до невдачі транзакції (є один виняток, який збільшує баланс рахунок).

У цій купі живе весь стан програми. Ваші облікові записи SOL, смарт-контракти та пам'ять, які використовуються смарт-контрактами. І кожна область пам'яті має програму, яка керує нею (іноді її називають «власником»). Термін solana для області пам'яті — «акаунт». Деякі програми володіють тисячами незалежних облікових записів. Ці облікові записи (навіть якщо вони належать одній програмі) не обов'язково мають бути однаковими за розміром.

2.3 Розробка програми

Для зручної розробки програм на блокчейні Solana був розроблений фреймворк Anchor з відкритим вихідним кодом, саме його ми будемо використовувати для розробки нашої програми.

Розробка буде виконуватися мовою програмування Rust.

Anchor[11] — це фреймворк для швидкого створення безпечних програм Solana.

За допомогою Anchor ми можете швидко створювати програми, оскільки він пише різні шаблони для нас, такі як (де)серіалізація облікових записів та даних інструкцій.

Ми можемо створювати безпечні програми легше, оскільки Anchor обробляє певні перевірки безпеки за нас. Крім того, це дозволяє коротко визначити додаткові перевірки та тримати їх окремо від нашої бізнес-логіки.

Обидва ці аспекти означають, що замість того, щоб працювати над виснажливими частинами необроблених програм Solana, ми можемо витратити більше часу на те, що найважливіше, на наш продукт.

Програма Anchor складається з трьох частин. Програмний модуль, структури Accounts, які позначені #[derive(Accounts)], і макрос declare_id. У програмному модулі ви пишете свою бізнес-логіку. У структурі Accounts ви перевіряєте облікові записи. Макрос declare_id створює поле ідентифікатора, яке зберігає адресу вашої програми. Anchor використовує цей жорстко закодований ідентифікатор для перевірок безпеки, а також дозволяє іншим крейтсам отримати доступ до адреси вашої програми[12].

2.3.1 Метод ініціалізації

Почнемо із створення методу ініціалізації нашої програми. Структура контексту, яку вона прийматиме, має такий вигляд (Рисунок 2.3.1).

```
#[derive(Accounts)]
pub struct StartOpenCharityTracker<'info> {
  #[account(init, payer = user, space = 8000)]
  pub base_account: Account<'info, BaseAccount>,
  #[account(mut)]
  pub user: Signer<'info>,
  pub system_program: Program <'info, System>,
}
```

Рисунок 2.3.1 - Структура контексту ініціалізації програми

Контекст має акаунт для запису туди транзакцій (base_account), акаунт власника та акаунт системної програми. base_account в свою чергу має наступну структуру (Рисунок 2.3.2). Структура має такі поля:

- owner, типу Pubkey, це поле зберігає публічний ключ акаунту який ініціалізував програму
- sum, типу f32, поле для зберігання суми зібраних коштів
- transactions, типу Vec<Transaction>, поле для зберігання масиву транзакцій

```
#[account]
pub struct BaseAccount {
    pub owner: Pubkey,
    pub sum: f32,
    pub transactions: Vec<TransactionStruct>,
}
```

Рисунок 2.3.2 - Струтура акаунту для збереження даних

Транзакції у свою чергу мають наступний тип даних(Рисунок 2.3.3). Структура має поле amount та hash.

- amount, типу f32, це поле з інформацією про кількість грошей переданих у транзакції
- hash, типу String, поле для хешу попередньої транзакції. Це поле потрібно для доказу нерозривності збирання коштів

```
#[derive(Debug, Clone, AnchorSerialize, AnchorDeserialize)]
pub struct TransactionStruct {
    pub amount: f32,
    pub hash: String,
}
```

Рисунок 2.3.3 - Струтура транзакції

З рисунку (Рисунок 2.3.2) ми бачимо, що на вхід метод приймає контекст, а на вихід повертае тип результату. Першим що робить метод, це ініціалізує дані, визначае початкову суму (що завжди дорівнює 0). Записує власника програми у base_account. Запам'ятовувати власника нам потрібно, щоб ніхто не міг додати свої транзакції у акаунт певного проекту по збору грошей.

```
use super::*;
pub fn start_open_charity_tracker(ctx: Context<StartOpenCharityTracker>) -> Result<()> {
    let base_account = &mut ctx.accounts.base_account;
    base_account.sum = 0.0;
    let owner = *ctx.accounts.user.key;
    base_account.owner = owner;
    Ok(())
}
```

Рисунок 2.3.4 - Код методу ініціалізації

2.3.2 Метод додавання транзакції

Тепер нам потрібно створити метод для додавання транзакції. Контекст методу додавання транзакції виглядає наступним чином(Рисунок 2.3.5)

```
#[derive(Accounts)]
pub struct AddTransaction<'info> {
    #[account(mut)]
    pub base_account: Account<'info, BaseAccount>,
    #[account(mut)]
    pub user: Signer<'info>,
}
```

Рисунок 2.3.5 - Контекст методу додавання транзакції

Контекст методу додавання транзакції приймає в себе акаунт для зберігання даних (base_account), та акаунт користувача хто володіє даними.

Метод додавання транзакції виглядає наступним чином(Рисунок 2.3.6)

```rust
pub fn add_transaction(ctx: Context<AddTransaction>, str_amount: String, hash: String) -> Result<()> {
    let owner = *ctx.accounts.user.key;
    let base_account = &mut ctx.accounts.base_account;
    let amount = str_amount.parse::<f32>().unwrap();
    if base_account.owner != owner {
        return Err(error!(ErrorCode::PermissionDenied));
    }

    let tsn = TransactionStruct {
        amount,
        hash: hash.to_string()
    };

    base_account.transactions.push(tsn);
    base_account.sum += amount;
    Ok(())
  }
}
```

Рисунок 2.3.6 - Метод додавання транзакції

На вхід метод отримує контекст, кількість грошей, та хеш. Першим що робить метод, це переводить кількість грошей з типу String до типу f32. Кількість грошей ми передаємо типом String через те, що десералізатори в солані не можуть поки що приймати числа з плаваючою точкою.

Далі перевіряємо, чи є той акаунт, який додає транзакцію, власником акаунта який зберігає транзакції. Якщо власник неправильний, метод повертає помилку. Структура помилок виглядає наступним чином(Рисунок 2.3.7).

```rust
#[error_code]
pub enum ErrorCode {
    #[msg("You are not the creator of the suggested account")]
    PermissionDenied,
}
```

Рисунок 2.3.7 - Структура помилок

Структура помилок має в собі лише одну помилку, яка буде повертатися лише у випадку описаному вище.

Далі метод приводить дані до правильної структури та зберігає у акаунті зберігання даних (base_account).

2.4 Тестування програми

Як було вказано вище, Anchor підтримує тести мовою програмування Typescript, саме тому ми будемо писати тести цією мовою програмування.

Перед тим як перейти до конкретних тестів ми повинні налаштувати середу тестування. Перед запуском тестів об'явимо провайдер, об'єкт програми, та згенеруємо base_account (Рисунок 2.4.1)

```
describe("opencharitytracker", () => {
  const provider = anchor.Provider.env();
  anchor.setProvider(provider);

  const program = anchor.workspace.Opencharitytracker as Program<Opencharitytracker>;

  const baseAccount = anchor.web3.Keypair.generate();
```

Рисунок 2.4.1 - Налаштування середи тестування

2.4.1 Тестування методу ініціалізації

Об'явимо тест для методу ініціалізації.

Перше що пробить тест, це викликає метод ініціалізації. Для виклику цього методу необхідно передати усі акаунти, що зазначені в контексті цього методу. Ми передаємо згенерований base_account, акаунт нашого провайдера, та системну програму. Далі ми повинні визначити хто буде підписувати транзакцію, як зазначено у контексті, це повинен бути base_account.

Після виконання цього методу ми повинні викликати метод fetch, для того щоб отримати інформацію про акаунт, ми передаємо публічний ключ base_account, та отримуємо всю актуальну інформацію про акаунт.

Тест вважається успішним якщо змінна sum, яка визначає суму зібраних коштів, дорівнює 0, тому у останньому рядку ми робимо саме цю перевірку.

```
it("Initialize", async () => {
  await program.methods.startOpenCharityTracker().accounts({
    baseAccount: baseAccount.publicKey,
    user: provider.wallet.publicKey,
    systemProgram: SystemProgram.programId,
  }).signers([baseAccount]).rpc();
  const account = await program.account.baseAccount.fetch(baseAccount.publicKey);
  assert.equal(account.sum, 0);
});
```

Рисунок 2.4.2 - Тест для методу ініціалізації

2.4.2 Тестування методу додавання транзакції

Об’явимо тест для методу додавання транзакції.

Перше що пробить тест, це викликає метод додавання транзакції. Для виклику цього методу необхідно передати усі акаунти, що зазначені в контексті цього методу. Ми передаємо згенерований та ініціалізований base_account, та акаунт користувача який володіє цим акаунтом. Також нам потрібно передати кількість грошей та деякий тест.

Спочатку ми викликаємо метод с транзакцією у якій кількість "10", та хеш - "some hash". Далі ми запитуємо у мережи усі дані про base_account. Перевіряємо чи дорівнює сума грошей 10ти. Якщо дорівнює, тоді частина тесту вважається виконеною.

Далі нам треба викликати ще раз додавання транзакції, викликаємо її зі значеннями "1.5" та "some hash". Аналогічно перевіряємо суму, вона повинна дорівнювати 11.5 (10 + 1.5). А також перевіряємо кількість транзакцій у масиві транзакцій, вона повинна дорівнювати 2. Якщо всі ці умови виконались, тест вважається виконаним.

```
it("Add transaction", async () => {
  await program.methods.addTransaction("10", "some hash").accounts({
      baseAccount: baseAccount.publicKey,
      user: provider.wallet.publicKey
    }).rpc();

  let account = await program.account.baseAccount.fetch(baseAccount.publicKey);

  assert.equal(account.sum, 10)

  await program.methods.addTransaction("1.5", "some hash").accounts({
      baseAccount: baseAccount.publicKey,
      user: provider.wallet.publicKey
  }).rpc();

  account = await program.account.baseAccount.fetch(baseAccount.publicKey);

  assert.equal(account.sum, 11.5)
  assert.equal((account.transactions as any[]).length, 2)
});
```

Рисунок 2.4.3 - Тест для методу додавання транзакції

РОЗДІЛ 3. РОЗРОБКА ГРАФІЧНОГО ВЕБ ІНТЕРФЕЙСУ

3.1 Огляд фреймворку для розробки

Розробку графічного веб інтерфейсу ми будемо робити мовою програмування Typescript, за допомогою фреймворка React.

React[13] (також відомий як React.js або ReactJS) — безкоштовна бібліотека JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача на основі компонентів інтерфейсу користувача. Його підтримує Meta (раніше Facebook), а також спільнота окремих розробників і компаній. React можна використовувати як основу для розробки односторінкових, мобільних або серверних додатків за допомогою таких фреймворків, як Next.js. Однак React лише керує станом і відтворює цей стан у DOM, тому створення додатків React часто вимагає додаткових бібліотек для маршрутизації та деяких функцій на стороні клієнта.

Основні переваги реакту перед іншими фреймворками:

- Декларативність: React робить безболісним створення інтерактивних інтерфейсів. Можна створити прості представлення для кожного стану у нашій програмі, і React ефективно оновлюватиме й відображатиме потрібні компоненти, коли наші дані змінюються. Декларативні уявлення роблять наш код більш передбачуваним, простішим для розуміння та легше налагоджуваним.
- На основі компонентів: можна створювати інкапсульовані компоненти, які керують своїм станом, а потім створювати їх, щоб створити складні компоненти інтерфейсу. Оскільки логіка компонентів написана на JavaScript, а не на шаблонах, ми можемо легко передавати розширені дані через свою програму і зберігати стан поза межами DOM.
- Навчайтеся один раз, пишіть будь-де: реакт робить припущень щодо решти вашого технологічного стеку, тому ми можемо розробляти нові функції в React, не переписуючи існуючий код. React також може

відображати на сервері за допомогою Node і використовувати мобільні додатки за допомогою React Native.

3.2 Створення дизайну у сервісі Figma

Figma — це векторний графічний редактор і інструмент для створення прототипів, який переважно працює в браузері з додатковими автономними функціями, які ввімкнено в настільних програмах для macOS та Windows. Мобільний додаток Figma для Android та iOS дозволяє переглядати та взаємодіяти з прототипами Figma на мобільних пристроях у режимі реального часу. Набір функцій Figma зосереджений на використанні в користувальницькому інтерфейсі та дизайні користувацького досвіду, з акцентом на співпрацю в реальному часі.

Для гарного користувацького досвіду нам потрібно відобразити інформацію про збір коштів, гафік для відображення статусу збору коштів, реквізити, а також список транзакцій які надійшли, біля кожної транзакції повинна бути посилання на екплорер блокчейну, де людина може переглянути транзакцію яку вона надіслала. У меню інформації про збір коштів повинна бути назва, на що збирають кошти, ціль - скільки грошей потрібно зібрати, та опис - декілька речень, більше інформації про те, на що будут збиратись кошти.

Дизайн інтерфейсу ви можете переглянути на рисунку (Рисунок 3.2.1).

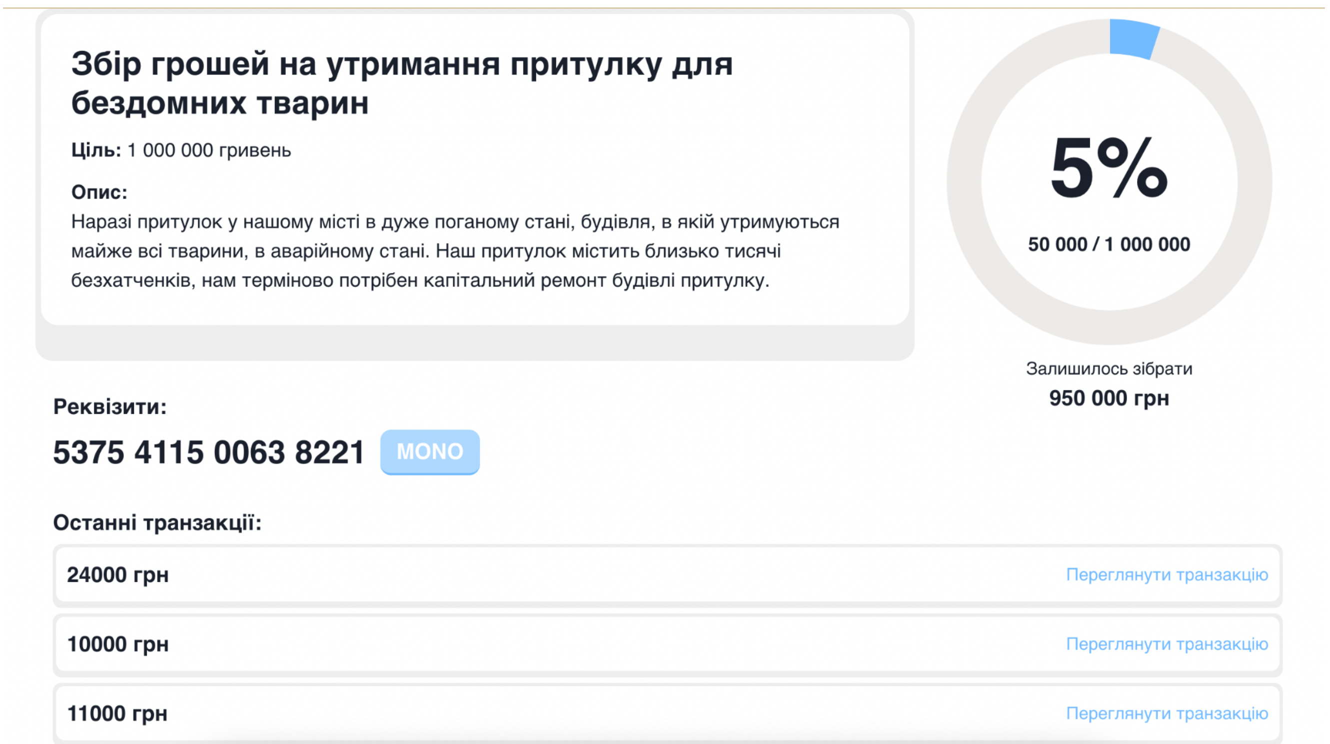


Рисунок 3.2.1 - Дизайн проекту

3.3 Розробка компонентів інтерфейсу

Поділимо сторінку 4 різні частини: інформація про проект, графік, реквізити, транзакції саме ці компоненти ми будемо розробляти. Для стилізування компонентів будемо використовувати бібліотеку ChackraUI.

3.3.1 Компонент інформації про проект

Компонент приймає дані, які зазначені в інтерфейсі IProps:

- name - назва проекту
- description - опис проекту
- goal - ціль по збиранню коштів

Після того реалізовуємо верстку проекту, кладемо всю інформацію у цей компонент, та завершуємо його написанням стилів для нього.

```
interface IProps {
  name: string;
  description: string;
  goal: number | string;
}

const ProjectInfo = ({ name, description, goal, ...rest }: IProps & BoxProps) => {
  return (
    <Box {...projectInfoStyle} {...rest}>
      <Box background="white" padding="28px" borderRadius="16px" height="100%">
        <Heading size="lg" mb="3">{name}</Heading>
        <Box fontSize="lg" mb="3"><b>Ціль:</b> {separateByThousands(goal)} гривень</Box>
        <Box fontSize="lg">
          <b>Опис: </b>
          <br />
          {description}
        </Box>
      </Box>
    </Box>
  );
};

const projectInfoStyle = {
  background: '#EEEEEE',
  border: '5px solid #EEEEEE',
  borderBottom: '33px solid #EEEEEE',
  borderRadius: '16px'
}

export default ProjectInfo;
```

Рисунок 3.3.1 - Код компонент інформації про проект

3.3.2 Компонент графік

Компонент приймає дані, які зазначені в інтерфейсі IProps:

- balance - скільки зібрано коштів
- goal - ціль по збиранню коштів

Після того реалізовуємо верстку проекту. Для коректної анімації поповнення, створюємо змінну prevBalance, за допомогою хука usePrevious. Далі рахуємо усі дані що зазначені у дизайні та створюємо графік. Графік реалізован за допомогою бібліотеки ChackraUI.

```
interface IProps {
  balance: number;
  goal: number;
}

const Chart = ({ balance, goal }: IProps) => {
  const prevBalance = usePrevious(balance);

  const percent = Math.round((balance * 100) / goal);
  return (
    <Flex direction="column" align="center">
      <CircularProgress size={320} value={percent} color="#72BBFF">
        <CircularProgressLabel>
          <Box fontWeight="bold">{percent}%</Box>
          <Box fontSize="lg" fontWeight="bold">
            {<CountUp separator=" " start={prevBalance || 0} end={balance} duration={0.5} />}
            /
            {separateByThousands(goal)}
          </Box>
        </CircularProgressLabel>
      </CircularProgress>
      <Flex flexDirection="column" justifyContent="center" alignItems="center">
        <Box> Залишилось зібрати </Box>
        <Box fontSize="xl" fontWeight="bold">
          <CountUp
            start={goal}
            end={goal - (prevBalance || balance)}
            duration={0.5} separator=" "
          />
          грн       You, 1 second ago • Uncommitted changes
        </Box>
      </Flex>
    </Flex>
  );
};

export default Chart;
```

Рисунок 3.3.2 - Код компоненту графік

3.3.3 Компонент реквізит

Компонент приймає дані, які зазначені в інтерфейсі IProps:

- address - сам реквізит
- label - деякий короткий опис, що за мережа реквізита

Після того реалізовуємо верстку проекту. Найбільша частина цього компоненту це стилі. Об’єкти стилів об’явлені у кінці компоненту.

```
interface IProps {
  address: string;
  label: string;
}

const Requisite = ({ address, label, ...rest }: IProps & BoxProps) => {

  return (
    <Box {...requisiteStyle} {...rest}>
        <Box fontSize="3xl" fontWeight="700">
          {formatCard(address)}
        </Box>
        <Box {...labelStyle}>
          {label}
        </Box>      You, now • Uncommitted changes
    </Box>
  );
};

const requisiteStyle = {
  display: 'flex',
  alignItems: 'center'
}

const labelStyle = {
  ml: 3,
  fontSize:"xl",
  fontWeight: "bold",
  color: "white",
  background: "#ADD8FF",
  padding: "5px 15px",
  borderRadius: "10px",
  borderBottom: "2px solid #72BBFF"
}

export default Requisite;
```

Рисунок 3.3.3 - Код компоненту реквізита

3.3.4 Компонент транзакції

Компонент приймає дані, які зазначені в інтерфейсі IProps:

- amount - кількість грошей передана транзакцією
- isLoading - значення яке визначає чи грузиться транзакція
- id - ідентифікатор транзакції

Після того реалізовуємо верстку проекту. Крім інформації про транзакцію, що нам передав сервер, ми також створюємо посилання на експлорер.

```
interface IProps {
  amount: string | number;
  isLoading?: boolean;
  id: string;
}
const Transaction = ({ amount, id, isLoading = false, ...rest }: IProps & BoxProps) => {
  return (
    <Box {...transactionStyle} {...{borderBottom: !isLoading ? "5px solid #EEEEEE" : ''}} {...rest}>
      <Box
        background="white"
        borderRadius="8px"
        padding="10px"
        display="flex"
        justifyContent="space-between"
        alignItems="center"
      >
        <Box fontWeight="bold" fontSize="xl">
          {amount} грн
        </Box>
        <Box as="a" href={`https://solscan.io/tx/${id}`} color="#72BBFF" cursor="pointer">
        Переглянути транзакцію
        </Box>
      </Box>
      {isLoading ?
        <Progress size='xs' colorScheme='blackAlpha' isIndeterminate borderRadius="16px"/>
      : null}
    </Box>
  );
};

const transactionStyle = {
  border: '2px solid #EEEEEE',
  padding: '1px',
  background: '#EEEEEE',
  borderRadius: "8px"
}

export default Transaction;
```

Рисунок 3.3.4 - Код компонента транзакції

3.4 Структури маніпулювання даними

Щоб правильно маніпулювати даними у фронтенді, було вирішено використовувати бібліотеку для управління станом додатка - Redux.

Redux[14] — це бібліотека JavaScript з відкритим кодом для керування та централізації стану програми. Він найчастіше використовується з такими бібліотеками, як React або Angular, для створення інтерфейсів користувача. Подібно (і натхненний) архітектурою Flux Facebook, він був створений Деном Абрамовим та Ендрю Кларком. З середини 2016 року основними супроводжуючими є Марк Еріксон і Тім Дорр.

Redux — це невелика бібліотека з простим обмеженим API, розробленим як передбачуваний контейнер для стану програми. Він працює подібно до функції скорочення, концепції функціонального програмування.

Але бібліотека Redux несе з собою багато рутинних дій, саме тому було вирішено використовувати бібліотеку Redux Toolkit як допоміжна до бібліотеки Redux.

Пакет Redux Toolkit[15] призначений як стандартний спосіб написання логіки Redux. Спочатку він був створений, щоб допомогти вирішити три поширені проблеми з Redux:

- "Налаштувати сторедж Redux занадто складно"
- "Мені потрібно додати багато пакетів, щоб Redux міг робити щось корисне"
- "Redux вимагає забагато шаблонного коду"

Бібліотека не може вирішити кожен варіант використання, але в дусі create-react-app вона може спробувати надати деякі інструменти, які абстрагуються від процесу налаштування та оброблятимуть найпоширеніші випадки використання, а також включати деякі корисні утиліти, які дозволити користувачеві спростити свій код програми.

Redux Toolkit також містить потужну можливість вибору даних і кешування, яку ми назвали «RTK Query». Він входить до комплекту як окремий набір точок входу.

Ці інструменти повинні бути корисними для всіх користувачів Redux. Незалежно від того, чи ви новий користувач Redux, який налаштовує свій перший проект, чи досвідчений користувач, який хоче спростити існуючу програму, Redux Toolkit може допомогти покращити код Redux.

3.4.1 Об'єкти станів

Для зберігання інформації про проект, я створив ось такий стейт (Рисунок 3.4.1).

```
const initialState: ProjectState = {
  transactions: [],
  requisites: { mono: "" },
  name: "",
  description: "",
  amount: 0,
  goal: 0,
};

export const projectSlice = createSlice({
  name: "project",
  initialState,
  reducers: {
    initProject: (state, { payload }) => {
      const { name, cards, description, goal, amount } = payload
      const mono = cards[0].number;
      return { ...state, name, description, goal, amount, requisites: { mono } };
    },
    addTransaction: (state, { payload }) => {
      const isTransactionExist = state.transactions.find(
        (tsn) => tsn.id === payload.id
      );
      if (!isTransactionExist) {
        state.transactions.push(payload);
      } else {
        const transactions = state.transactions.map((tsn) =>
          tsn.id === payload.id ? payload : tsn
        );
        return { ...state, transactions };
      }
    },
    loadTransaction: (state, { payload }) => {
      const transactions = state.transactions.map((tsn) => {
        return tsn.id === payload.id ? { ...tsn, isLoading: false } : tsn;
      });
      return { ...state, transactions };
    },
  },
});
```

Рисунок 3.4.1 - Стейт проекту

Цей стейт має початковий стан який оголошений у об'єкті initialState, а також деякі екшени. Вони визначені задля додавання даних до стейту. Саме у цьому прикладі є такі екшени:

- initProject - екшен задля вставновки стартової інформації про проект
- addTransaction - додавання транзакції у список транзакції
- loadTransaction - змін стану транзакції з завантаження до завантаженої транзакції

Також був створений стейт для приєднання веб-сокетів до серверу, ось так он виглядає (Рисунок 3.4.2)

```
interface WsState {
  socket: Socket;
}
const socket = io(`${env.api}`);

const initialState: WsState = {
  socket,
};

export const wsSlice = createSlice({
  name: "ws",
  initialState,
  reducers: {},
});

export default wsSlice.reducer;
```

Рисунок 3.4.2 - Код приєднання веб-сокетів

3.4.2 Об'єкти роботи з запитами

Також потрібно створити об'єкт у якому ми будемо працювати з запитами. Робити ми будемо лише один запит, який буде брати усі дані про проект.

```
export const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: env.api }),
  endpoints: (builder) => ({
    getInfo: builder.query({
      query: (id) => `project/${id}`,
      async onQueryStarted(_req, { dispatch, queryFulfilled }) {
        let { data } = await queryFulfilled;
        dispatch(initProject(data));
      },
    }),
  }),
})

export const { useGetInfoQuery, useLazyGetInfoQuery } = api
```

Рисунок 3.4.2 - Об’єкт для роботи з запитами

3.4.3 Веб-сокет хендлери

Для того, щоб транзакції на сторінці обновлювалися без перезавантаження сторінки, було вирішено використовувати веб-сокети. На рисунку(Рисунок 3.4.3) можна побачити як реалізовані хендлери веб-сокет відповідей. Вони змінюють стейт в залежності від відповіді

```
const onTransactionInitiated =
  (dispatch: Dispatch) =>
  ({ amount, id }: { amount: string; id: string }) => {
    dispatch(addTransaction({ amount, id, isLoading: true }));
  };

const onTransactionFinalized =
  (dispatch: Dispatch) =>
  ({ id }: { id: string }) => {
    dispatch(loadTransaction({ id }));
  };

export const transactionHandler = (socket: Socket, dispatch: Dispatch) => {
  socket.on(Events.TransactionInitiated, onTransactionInitiated(dispatch));
  socket.on(Events.TransactionFinalized, onTransactionFinalized(dispatch));
};

export const transactionHandlerDisconnect = (socket: Socket) => {
  socket.off(Events.TransactionInitiated);
  socket.off(Events.TransactionFinalized);
};
```

Рисунок 3.4.3 - Код хендлерів веб-сокет відповідей

3.5 Контейнер головної сторінки

У контейнері головної сторінки викликається запит на отримання інформації про проект. Після того як дані завантажились вони збираються зі стейту і далі відображаються у верстці. З рисунка (Рисунок 3.5.1) можна побачити, що ми використовуємо усі компоненти, що були описані вище.

```
const Home = () => {
  const {
    isError, isLoading
  } = useGetInfoQuery('f3e4c107-225e-42f6-a20c-8db51c1213ae');

  const { transactions, name, description, goal, amount, requisites } = useAppSelector(selectProject);

  if (isError) return <div>An error has occurred!</div>
  if (isLoading) return (
    <Flex justifyContent="center" alignItems="center" mt={5}>
      <CircularProgress size={320} isIndeterminate color="#72BBFF" />
    </Flex>
  )

  return (
    <Box width="80%" margin="0 auto">
      <Flex direction="row" justify="space-around" mb="30px">
        <Flex direction="column">
          <ProjectInfo
            name={name}
            description={description}
            goal={goal}
            mb="30px"
            mr="5"
          />
          <Box ml="16px">
            <Heading size="md" mb={2}>Реквізити: </Heading>
            <Requisite address={requisites.mono} label="MONO" />
          </Box>
        </Flex>
        <Chart balance={amount} goal={goal} />
      </Flex>
      <Flex direction="column" ml="16px">
      <Heading size="md" mb={2}>Останні транзакції: </Heading>
        {(transactions || []).map((tsn: any) => (
          <Transaction {...tsn} marginBottom="5px" />
        ))}
      </Flex>
    </Box>
  );
};

export default Home;
```

Рисунок 3.5.1 - Код контейнеру головної сторінки

ВИСНОВКИ

Зараз, як і будь-коли, гостро стоїть питання довіри до людей, які збирають гроші. З розвитком економіки розвиваються як способи збирання грошей, так і способи шахрайства. Але завдяки розвитку комп'ютерних наук ми можемо допомогти боротися з шахрайством. Ми знаємо, що не всі збирачі грошей – чесні. Часто благодійні компанії причетні до корупційних схем, бо часто позбавлені сплати податків. Та й просто, навіть люди на вулиці, які збирають гроші, дуже часто спустошують свій капелюх, щоб усім здавалося, що їм ще ніхто не дав грошей, таким чином вони тиснуть на жаліcть. Всі цивілізовані держави допомагають чесним людям збирати гроші на певні цілі, а мета цієї роботи була в тому, щоб допомогти всім простим людям уникнути шахраїв і взяти на себе невелику відповідальність у наданні даних про кількість коштів на рахунках людей, які організовують збори грошей. Я розробив графічний інтерфейс та блокчейн смарт контракт для проекту, який відображає кількість грошей на банківському рахунку. Всі транзакції, що приходять на банківський рахунок, зберігаються в блокчейні, відповідно не можуть бути видалені, і завжди будуть доступні. З цією інформацією людина завжди може її взяти і звернутися до суду, заявивши, що людина чи організація, яка збирає гроші - шахрай.

Частини, які я розробив, повністю протестовані і стабільно працюють. Через те, що блокчейн, який використовує проект, має дуже хорошу пропускну спроможність, обслуговування його програми коштуватиме менше цента за кожну транзакцію. Так як на ринку немає аналогів такого проекту, я вважаю, що він може бути дуже перспективним.

ДОДАТКИ

https://github.com/AlexanderTirik/OpenCharityTracker - посилання на гітхаб репозиторій виконаного проекту

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Словник Merriam-webster. Визначення слова Blockchain [Електронни ресурс] // URL: https://www.merriam-webster.com/dictionary/blockchain
2. Оксфордський словник. Визначення слова Blockchain [Електронний ресурс] // URL: https://www.lexico.com/definition/blockchain
3. Investopedia, Blockhain Explained. By ADAM HAYES, Reviewed by JEFREDA R. BROWN, Fact checked by SUZANNE KVILHAUG [Електронний ресурс] // URL: https://www.investopedia.com/terms/b/blockchain.asp
4. Github - bitcoin/bitcoin [Електронний ресурс] // URL: https://github.com/bitcoin/bitcoin
5. Bitcoin Core [Електронний ресурс] // URL: https://bitcoincore.org/
6. Github – ethereum [Електронний ресурс] // URL: https://github.com/ethereum
7. Офіційний сайт блокчейну Ethereum [Електронний ресурс] // URL: https://ethereum.org/uk/
8. Офіційний сайт блокчейну Solana [Електронний ресурс] // URL: https://solana.com/en
9. Офіційна документація блокчейну Solana [Електронний ресурс] // URL: https://docs.solana.com/
10. Github - solana-labs [Електронний ресурс] // URL: https://github.com/solana-labs/
11. Github - project-serum/anchor [Електронний ресурс]// URL: https://github.com/project-serum/anchor
12. Документація фреймворку Anchor [Електронний ресурс]// URL: https://book.anchor-lang.com/
13. Офіційний сайт фреймворку React [Електронний ресурс]// URL: https://uk.reactjs.org/
14. Офіційний сайт бібліотеки Redux [Електронний ресурс]// URL: https://redux.js.org/

15.Офіційний сайт бібліотеки Redux-Toolkit [Електронний ресурс]// URL: https://redux-toolkit.js.org/

16. On Optimistic Methods for Concurrency Control. H.T.Kung, J.T.Robinson (1981)