Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____Н.В. Лукова-Чуйко
« » червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи

бакалавра

(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Блокчейн в сфері кібербезпеки і логістики»

Виконавець: студент IV курсу, групи КБ-41
Фаль Лука Хассенович

(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Шестак Я.В.	

Нормоконтроль	Зюбіна Р. В.	

Київ 2021

Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:

завідувач кафедри кібербезпеки

та захисту інформації

_____Н.В. Лукова-Чуйко

«10» жовтня 2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-41** **Фаль Лука Хассенович**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Блокчейн в сфері кібербезпеки і логістики

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Концепція блокчейну, методи шифру та робота логістики.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно ознайомитися з теорією блокчейн технологій, вразливостями з боку безпеки, обрати методи шифрування, та ознайомитися з теорією логістики.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблена система безпеки яка може позбавитись від

корупції та підробки в сферах доставках

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видав		Я. В. Шестак
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		Л. Х. Фаль
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.01.2021 – 29.01.2021	*виконано*
2	Аналіз літератури	29.01.2021 – 14.02.2021	*виконано*
3	Обґрунтування вибору рішення	15.02.2021 – 18.02.2021	*виконано*
4	Концепція блокчейн технологій	19.02.2021 – 05.03.2021	*виконано*
5	Аналіз проблем логістики та кибер безпеки в сферах доставки	06.03.2021 – 19.03.2021	*виконано*
6	Вироблення системи для передачі даних до блокчейну і методи захисту даних	20.03.2021 – 12.04.2021	*виконано*
7	Вироблення системи зчитування даних з RFID міток	13.04.2021 – 8.05.2021	*виконано*
8	Оформлення пояснювальної записки	18.05.2021 – 08.06.2021	*виконано*
9	Підготовка до захисту дипломної роботи	09.05.2020 – 21.06.2021	*виконано*

Завдання видав		Я. В. Шестак
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		Л. Х. Фаль
	(підпис)	(ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 60 сторінок основного тексту, 3 таблиці та 1 формулу. Список використаних джерел містить 51 найменування і займає 5 сторінок.

Метою даної роботи є оцінка поточного стану захищеності логістичних сервісів, та розробка системи щодо захисту товару та доставок за допомогою блокчейн технології.

У роботі проаналізована існуюча література з теорії блокчейн технології, виконаний аналіз документів, порівняння, вивчення та узагальнення вітчизняної і зарубіжної практики з теми блокчейн технологій, розроблена система блокчейн логістики.

Розроблений лістинг програми, що виконує шифрування SHA-256, може використовуватися користувачами для створення приватного ключа для доступу до блокчейн системи.

Розроблені рекомендації призначені для користувачів, що хочуть забезпечити безпеку своїх персональних даних в хмарних сервісах.

Ключові слова: Блокчейн технології, RFID мітки, логістика, приватний ключ, відкритий ключ, приватна блокчейн система

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES	–	Advanced Encryption Standard
API	–	Application Programming Interface
SHA-256	–	Secure Hash Algorithm 256 bit
RFID	–	Radio Frequency Identification
IoT	–	Internet-of-Things
IT	–	Information Technology
SSL	–	Secure Sockets Layer
SPI	–	Serial Peripheral Interface
VPN	–	Virtual Private Network
ПЗ	–	Програмне забезпечення
DCCD	–	Decentralized cargo control delivery
BlockChain	–	Вузол ланцюжок

ЗМІСТ

ВСТУП

Міжнародні контейнерні перевезення - це одна з найбільш забюрократизованих галузей в сучасній економіці, так як в процесі доставки бере участь безліч країн, державних структур і компаній, в кожній з яких свої правила роботи з документами і системи обліку. Для прикладу, доставка рефрижераторних вантажів в Європу зі Східної Африки вимагає штампів і дозволів приблизно від 30 людей і організацій, які повинні взаємодіяти один з одним більш ніж в 200 випадках. Сукупні витрати на обробку документів при такій логістиці оцінюються в межах від 15 до 50% від вартості фізичного транспорту.

Згідно з оцінкою Всесвітнього економічного форуму, зниження бюрократичних бар'єрів для торгівлі в логістиці та ланцюжках поставок збільшить обсяг світового ВВП на 5%, глобальної торгівлі - на 15%. При цьому, на думку фахівців, блокчейн - єдина технологія, яка може прибрати ці бар'єри.

Актуальність теми дипломна робота зумовлена важливістю дослідження та розробки рішень, що допоможуть контролювати як локальні так і глобальні ланцюги поставок будь-яких вантажів. Дані кожної одиниці вантажу слід зберігати в математично та криптографічно захищеній системі, що допоможе позбутися бюрократії.

Аналіз останніх досліджень та літератури. Вчені, які зробили вклад у вивчення блокчейн технологій: Натаниэль Поппер, Алекс Форк, Дон Тапскотт, Алекс Тапскотт, Мелани Сван, Кирк Дэвид Филипс.

Мета дипломної роботи є побудова програмної системи для можливості контролю автентичності вантажу за допомогою блокчейну та RFID мітки. Для цього необхідно вирішити такі завдання:

- Дослідити теоретичні основи побудови permissioned блокчейну.
- Ознайомитись з видами RFID-міток та особливостями роботи з ними.
- Розробити прототип permissioned блокчейну та описати API для доступу до даних в ньому.

- Розробити прототип RFID-читача та реалізувати можливість відправки даних з RFID-мітки до користувача.

- Розробити користувацький інтерфейс у вигляді SPA (single page application) для доступу до блокчейну, використовуючи реалізований API.

Об'єктом дослідження є децентралізована та криптографічно захищена система для контролю логістичних ланцюгів зі зв'язком з реальним світом за допомогою RFID-чіпів, основні підходи її реалізації.

Предметом дослідження дипломної роботи є: концепція побудови децентралізованих систем та блокчейну зокрема, програмно-технічні підходи до розробки рішень для зв'язку інформації, що зберігається в RFID-мітках, з блокчейном, архітектурні підходи до побудови якісних API для можливості розробки користувацьких інтерфейсів різного плану, особливості та основні принципи розробки користувацьких інтерфейсів та SPA (single-page application) зокрема.

У процесі виконання дипломної роботи було використано: Raspberry Pi 4 Model B та документація до неї, радіомітка ISO14443A NTAG 213 та документація до неї, програмне забезпечення Visual Studio code, документація React, JavaScript, Python, міжнародні та національні стандарти з побудови програмних систем, стандарти з побудов приватних блокчейнів та користувацьких інтерфейсів.

Методи дослідження дипломної роботи:

- аналіз літератури;
- аналіз документів;
- порівняння;
- вивчення та узагальнення вітчизняної і зарубіжної практики.

РОЗДІЛ 1
АНАЛІЗ ГАЛУЗІ ЛОГІСТИКИ.АНАЛІЗ ІТ ЗАБЕЗПЕЧЕННЯ ЛОГІСТИЧНОЇ СФЕРИ В УКРАЇНІ. АНАЛІЗ НАЯВНИХ СВІТОВИХ РІШЕНЬ В СФЕРІ ЛОГІСТИКИ. ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз сучасного стану галузі логістики

Наразі ланцюги поставок стали настільки складні, що пересічний споживач не має можливості відстежити походження товару і підтвердити його справжність. Як підсумок, фармацевтична і продуктова галузь загрузли в скандалах і суднових позовах через підроблену, контрафактну або зіпсовану продукцію. Лише за офіційними даними Інтерполу і ВООЗ, 1 мільйон осіб щорічно помирає від контрафактних препаратів. Згідно з дослідженнями, 1% ліків в розвинених країнах і 10% ліків в світі - підробка. Ще 8,5% всієї фармацевтичної продукції просто псується через неналежні умови перевезення.

За допомогою суміші технологій Інтернету речей та блокчейну цю статистику можна знизити до мінімуму. Кожен раз, коли товар переходить з рук в руки, ця транзакція прописується в блокчейні, що створює безперервну історію руху продукту по ланцюжку продажів від виробника до полиці в супермаркеті або аптеці. Завдяки цьому можна вирішити такі завдання:

- запис моменту передачі активів і їх кількості - таких, як контейнери, піддони, трейлери - при переміщенні товару по ланцюжку продажів;

- відстеження замовлень на продаж або купівлю, відправка і зміна повідомлень про доставку, квитанцій, договорів або інших пов'язаних з вантажоперевезенням документів;

- привласнення певних властивостей продуктів або перевірка сертифікатів, наприклад, перевірка справжності походження продукту або того, чи є він органічним;

- зв'язування фізичних товарів, контейнерів, транспорту і обладнання з серійними номерами, цифровими мітками, штрих-кодами за допомогою RFID-міток (радіочастотний ідентифікатор);

- обмін інформацією про процес збору продуктів, виробництв, доставці, фінансових розрахунках і обслуговуванні продуктів.

«Інтернет речей», або IoT (скорочення від англійського - «Internet of Things») - це все повсякденні об'єкти, які можуть обмінюватися даними через інтернет. У дослідницької та консалтингової компанії Gartner підрахували, що до 2020 року в світі буде більше 20 млрд IoT-пристроїв. Логістика - одне з найбільш перспективних напрямків для впровадження IoT. Наприклад, при доставці контейнера він зможе самостійно підтверджувати доставку, вказуючи час і стан вантажу. Потім система автоматично перевірить, чи був товар доставлений відповідно до погоджених умов (температура, вологість, нахил), і випустить платежі відповідним сторонам.

В Україні в галузі логістики ідея автоматизація бізнес процесів за допомогою RFID міток з кожним роком стає все більш популярною. Але дані все рівно зберігаються в централізованих базах даних, безпека доступу до яких залишає бажати кращого.

1.2 Аналіз IT забезпечення логістичних послуг в Україні

Було проаналізовано дві ключові організації в сфері поштових послуг в Україні: «Укрпошта» та «Нова пошта» та зроблені відповідні висновки відносно стану IT забезпечення логістичної сфери країни.

Акціонерне товариство «Укрпошта» (АТ «Укрпошта») — державне підприємство поштового зв'язку, підпорядковане Міністерству інфраструктури України, національний оператор поштового зв'язку. Підприємство забезпечує надання універсальних послуг поштового зв'язку, перелік яких і відповідні тарифи затверджуються Національною комісією з питань регулювання зв'язку, а також має виключне право на видання, введення в обіг та організацію розповсюдження поштових марок, маркованих конвертів і карток, а також виведення їх з обігу в

Україні. В основному за останні роки були зроблені істотні зміни зі сторони інформатизації різного роду бізнес процесів. Та наразі є досить перспективні плани на майбутнє. В травні 2020 року генеральний директор Укрпошти Ігор Смілянський розповів журналістам, що до осені 2021 року в "Укрпошті" не повинно залишитися жодних паперових операцій. Пошта стане повністю диджиталізована. Зокрема, за словами начальника поштового відомства, після закупівлі нової системи Front Office кожна листоноша пересувного відділення буде оснащена спеціальним мобільним пристроєм з технологією NFC (технологія, яка об'єднує між собою кілька пристроїв для бездротової передачі даних. Інтерфейс, що використовується в більшості сучасних смартфонів і планшетів). Окрім фінансових операцій, такі пристрої можуть також видавати чеки.

«Нова пошта» поштова логістична компанія, заснована у 2001 році. Спеціалізується на експрес-доставці вантажів. У 2019 році компанія увійшла до переліку найбільших в Україні платників податків. «Нова Пошта» планує інвестувати близько $100 млн у розвиток компанії 2020 року. Цьогоріч буде запущено другу чергу КІТа (Київський інноваційний центр – Mind), підвищивши потужність до 50 000 з перспективою збільшити до 60 000 посилок на годину.

Також компанія планує побудувати чотири великі інноваційні центри у місті Бровари, у Дніпрі, в Одесі та Харкові з автоматизованою сортувальною системою. Потужності кожного складуть мають зрости до 20 000 посилок на годину. Крім того в плани «Нової Пошти» входить побудова першого вантажного терміналу з роботизованою системою сортування палет та великогабаритних вантажів. Також мають бути модернізовані відділення «Нової Пошти» та запроваджено нове обладнання, щоб забезпечити клієнтам досконалий сервіс та швидкість обслуговування. Буде проведено закупівлю скутерів, пікапів, бусів та міжміських контейнерів. «Звісно, велика частина інвестицій піде на IT та розвиток мобільних сервісів. Також запустимо нові продукти, про які буду розповідати окремо. Зробимо все, щоб наші улюблені клієнти отримували найкращий сервіс, швидкість та якість доставки. Поки всі чекають інвестицій з Давосу, внутрішні інвестори вже вкладають

в Україну», – розповів журналістам співзасновник компанії Володимир Поперешнюк.

Все виглядає досить перспективно з точки зору швидкості, зручності, глобалізації та збільшення прибутку. Але перевезення з використанням таких технологій ніколи не можуть забезпечити ні безпеку даних клієнтів, ні підтвердити належні умови доставки важливих вантажів. В мережі все частіше можна зустріти бази даних з персональними даними користувачів. «Нова пошта» взагалі на коробках з вантажем прикріплює аркуш, на якому вказано ФІО та мобільний номер телефону відправника та отримувача. Ці дані можуть бути збережено на кожному етапі доставки зацікавленими особами. Такі технології не в силах гарантувати відсутність фальсифікації, підробки та заміни вантажу. Користувач може бачити лише знаходження його вантажу на якомусь етапі перевезення, але це не гарантує абсолютно нічого. Це велика лотерея, тому приватні компанії, які виготовляють товари з критично важливими умовами перевезення, використовують власні логістичні сполучення. Якщо говорити про Україну, то насамперед саме таким компаніям і потрібно впровадження блокчейну з використанням Інтернету речей. Адже підприємство, наприклад, яке виготовляє дорогі ліки з критичною вразливістю до високих температур, хоче контролювати кожний крок на етапі доставки вантажу та при цьому гарантувати його справжність. Наявні технології в глобальній логістичній інфраструктурі Україні не в змозі цього гарантувати. Впровадження блокчейну та Інтернету речей допоможе не тільки контролювати умови перевезення вантажу на кожному кроці його доставки, а й позбутися потреби розкривати персональні дані.

1.3 Аналіз наявних світових рішень в сфері логістики

Maersk та IBM об'єдналися для створення глобальної системи для оцифровки торгових операцій і контролю вантажоперевезень – TradeLens. Їх система дозволить кожній зацікавленій особі стежити за рухом товарів по ланцюжку поставок, щоб розуміти, що з контейнером і де він знаходиться в реальному часі.Зацікавлені

сторони також можуть бачити статус митних документів, переглядати коносаменти та інші відомості. Технологія блокчейн забезпечить безпечний обмін інформацією і сховище для документації, захищене від несанкціонованого втручання.

DHL та Accenture запустили проект серіалізації на основі блокчейну (рис. 1.1), що надає всім зацікавленим особам можливість відстеження і перевірки автентичності фармацевтичної продукції.

Рисунок 1.1 Схема серіалізації за проектом DHL і Accenture

Серіалізація - це процес присвоєння унікального ідентифікатора (наприклад, серійного номера або QR-коду) для кожного герметизованого контейнера, який потім пов'язують з важливою інформацією про походження товару: виробник, номер партії, дата закінчення терміну дії, дані про перевезення. Це дозволить відстежити місце розташування товару на будь-якому етапі його життєвого циклу і підтвердити його справжність.

Під час написання дипломної роботи, на початку березня 2021 року бренди Nike, PVH Corp (Tommy Hilfiger, Calvin Klein), HermanKay і роздрібні продавці Kohl

і Macy's почали використовувати блокчейн для контролю поставок одягу за допомогою RFID-міток.

Компанії з виробництва одягу постійно стикаються з загрозою контрафактної продукції, піратства та крадіжки. Блокчейн здатний забезпечити прозорість ланцюжка поставок і контрольований потік даних про місцезнаходження продукту. Лабораторія RFID Lab при Університету Оберна (Auburn University RFID Lab) розробила блокчейн-платформу CHIP (CHainInegration Project), яка призначена для контролю поставок продукції від місця виробництва до магазину. Згадані компанії брали участь у першому етапі проекту, який успішно інтегрував потоки даних про продукцію з різних вузлів ланцюжка поставок з січня по грудень 2019 року.

1.4 Постановка технічного завдання для дипломної роботи

Основне завдання дипломної роботи є розробка прототипу системи для контролю різного роду ланцюгів поставок в галузі логістики. Для цього можна використати уже готовий блокчейн, наприклад, Hyperledger від LinuxFoundation. Hyperledger - це спільнота з відкритим кодом, орієнтована на розробку набору стабільних рамок, інструментів та бібліотек для розгортання приватного блокчейну для корпоративного бізнесу. Для розробки уже реального проекту я б використовував саме цей інструментарій, але в цілях глибокого дослідження, в першу чергу, технології блокчейн, в рамках цієї дипломної роботи я буду створювати прототип приватного блокчейну власноруч, використовуючи мову програмування Python та бібліотеку Flask. Для цього потрібно буде дослідити математичне та криптографічне підґрунтя цієї технології, основні принципи розробки системи та особливості проектування API.

Другою важливою частиною дипломної роботи є фізична сторона – Інтернет речей. В образі цієї технології буде виступати RFID мітка ISO14443A NTAG 213. Вона буде зберігати унікальний ідентифікатор товару та передавати його користувачу в процесі зчитування спеціальним читачем. Для розробки читача я використовую RaspberryPi 4 Model B та за допомогою протоколу SPI розробляю

драйвер для MFRC522 RFID читачу. За допомогою MFRC522 користувач зможе зчитати інформацію з RFID мітки та використати її або для створення нової транзакції, або для пошуку даних про цей товар. В реальному проекті слід використовувати RFID мітки, що мають змогу проводити авторизацію читачів, таким чином відсіюючі зловмисників, але в рамках цієї дипломної роботи буде використана звичайна RFID мітка і як ідентифікатор вантажу буде використовуватися її унікальний ідентифікатор.

Третьою частиною дипломної роботи являється користувацький інтерфейс. Використовуючи API розробленого блокчейну я побудую сайт за допомогою фреймворк ReactJS. На ньому користувач може досліджувати блокчейн та створювати нові транзакції.

Висновки за розділом 1

У цьому розділі розглянули поняття блокчейн системи. Вивчили методи зчитування RFID міток, проаналізували ринок логістики в Україні та зарубіжом. Крім цього, ми описали основні цілі захисту інформації у системах блокчейну з клієнтом. Також визначили основні загрози під час використання блокчейн систем

Після всього цього ми вирішили як ми будемо будувати наш веб-додаток і описали чому ми обрали саме його. Отже, результатом дослідження у цьому розділі стали описані загрози, ринки, веб-додатки, блокчейн технологій, RFID мітки та считувальні процеси.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СТРУКТУРИ СИСТЕМИ КОНТРОЛЮ ЯКОСТІ ДОСТАВКИ ВАНТАЖУ. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРИВАТНОГО БЛОКЧЕЙНУ, ЙОГО API, ЗВ'ЯЗКУ RFID МІТКИ З БЛОКЧЕЙНОМ ТА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

2.1 Проектування структури системи

Основою прототипу системи контролю якості вантажу буде блокчейн, що буде виконувати функцію децентралізованої бази даних та буде контролювати та верифікувати будь-які транзакції у всій системі. RFID мітка буде збирати та зберігати інформацію про товар та його умови перевезення. RFID-читач буде виконувати функцію ретранслятора даних з RFID мітки до блокчейну. Кожний RFID-читач матиме авторизований приватний ключ та матиме змогу самостійно формувати готову транзакцію та відправляти її до блокчейну без втручання користувача.

Верифікація читачів відбувається на етапі встановлення блокчейну на логістичний ланцюг підприємства. При необхідності видалення чи добавлення нових авторизованих читачів буде створюватися смарт-контракт. Смарт-виглядатиме як звичайна транзакція, але потребуватиме підтвердження від кожного авторизованого вузла в системі. Після верифікації смарт-контракта буде перезаписано відповідний файл з авторизованими валидаторами. Кінцевий користувач вантажу матиме змогу дізнатися про умови доставки та точний час знаходження вантажу на контрольних пунктах через користувацький інтерфейс у вигляді сайту.

2.2 Розробка приватного блокчейну

Блокчейн, тобто ланцюжок блоків транзакцій (англ. Blockchain, Blockchain Від block — блок, chain — ланцюг) — розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), що постійно довшає. Дані захищено від підробки та спотворення. Кожен блок містить часову позначку, хеш попереднього блока та дані транзакцій, подані як хеш-дерево. Таку розподілену базу даних покладено в основу криптовалюти Bitcoin (вона була описана 2008 і реалізована 2009 року), де слугує бухгалтерською книгою для всіх операцій. Таку базу називають Блокчейн.

Блокчейн називається приватним, коли право на валідацію, створення нових блоків та транзакцій має обмежена кількість осіб. В нашому випадку це будуть особи, що братимуть безпосередній контакт з вантажем на контрольних точках.

2.2.1 Основні об’єкти розробленого блокчейну та їх функціонал

ClassTransaction. Клас який зберігає всі дані про вантаж. Транзакція може створюватися автоматично при зчитуванні RFID-читачем на контрольній точці. При цьому в блокчейн потрапляє вся актуальна інформація вантажу з цифровим підписом читача. Поля класу:

- timestamp – час створення транзакції у форматі POSIX**.**
- information – бізнес інформація про вантаж. Задається на першій контрольній точці та існує в єдиному екземплярі - в першій транзакції з вантажем.
- temperature – температура повітря на контрольній точці.
- cargo_id – унікальний ідентифікатор вантажу. За допомогою нього можна буде виконувати пошук усіх наявних транзакцій в блокчейні за допомогою користувацького API.
- hash – хеш транзакції, що отримується шляхом конкатенації всіх полів класу та отримання з цих даних sha256 дайджесту.

- public_key – публічний ключ власника створеної транзакції. Використовується для перевірки коректності цифрового підпису.

- signed hash – хеш транзакції, що підписаний приватним ключем відправника.

ClassBlock. Грає безпосередню роль у створенні блокчейну. Зберігає певні транзакції, що були в нього добавлені вузлом після валідації. Зберігає в собі всі дані для перевірки коректності його даних та його розташування в блокчейні.

- timestamp – час створення блоку у форматі POSIX**.**

- transactions – масив транзакцій, з яких був зібраний блок. Щоб створити блок потрібна хоча б 1 транзакція в pendingpool (транзакції, що очікують валідації). Про memory pool буде написано далі в цьому підрозділі.

- merkle_root – хеш, що зберігає інформацію про всі транзакції, що були додані в блок, коли він створювався. За допомогою цього алгоритму можна з точністю до транзакції сказати, де відбулася зміна. Алгоритм створення MerkleRoot буде описано далі в цьому підрозділі.

- previous_hash – хеш попереднього блоку. Використовується для валідації блокчейну. За рахунок цієї інформаційне власне і існує блокчейн. Будь-який користувач або вузл може перерахувати хеші усіх блоків та з точністю до транзакції виявити похибку.

- hash – хеш блоку, що отримується шляхом конкатенації всіх полів класу та отримання з цих даних sha256 дайджесту.

- public_key – публічний ключ власника створеної транзакції. Використовується для перевірки коректності цифрового підпису.

- signed_hash – хеш транзакції, що підписаний приватним ключем відправника.

ClassBlockchain. Цей клас зберігає всі корисні дані для поточного вузла мережі блокчейн. Поля класу:

- chain – зберігає масив всіх блоків, що відомі поточному вузлу.

- friendly_nodes - масив url-посилань дружніх вузлів в мережі. За допомогою них буде виконуватися децентралізація даних, верифікація всіх вхідних транзакцій та реалізована черга для створення нових блоків.

- miner_queue_number – номер в черзі створення блоків для поточного вузла. Алгоритм визначення черги вузла буде описано далі в цьому підрозділі.

2.2.2 Memorypool

Memorypool або pendingpool – дуже важлива частина системи. В ній зберігаються всі транзакції, що створюються на контрольних точках. В момент створення транзакції, вона не попадає відразу до блокчейну, спочатку вона попадає до memorypool, де очікує валідації та потрапляння до блоку. Коли вузол отримує новую транзакцію він добавив її в memorypool та надсилає її всім дружнім вузлам мережі. Якщо транзакція не проходить валідацію, або приходить блок від іншого вузла, який містить цю транзакцію, то тоді дана транзакція видаляється з memorypool поточного вузла.

2.2.3 Merkleroot

Хеш-деревом, деревом Меркле (англ. Merkletree) називають повне бінарне дерево, в листові вершини якого поміщені хеші від блоків даних, а внутрішні вершини містять хеш-кодування від додавання значень в дочірніх вершинах. Кореневий вузол дерева містить хеш від усього набору даних, тобто хеш-дерево є односпрямованої хеш-функцією (рис. 2.1.). Дерево Меркла застосовується для ефективного зберігання транзакции в блокчейні криптовалют (наприклад, в Bitcoin, Ethereum). Воно дозволяє отримати «відбиток» всіх транзакцій в блоці, а також ефективно верифікувати транзакції.

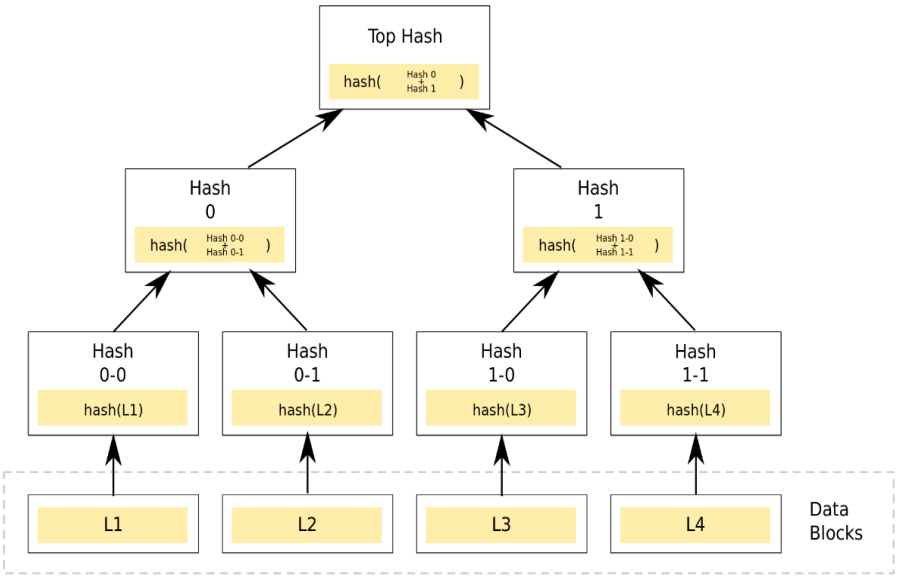


Рисунок 2.1 Алгоритм бінарного хеш-дерева Меркла

Для зберігання кореня хеш-дерева Меркла (Merkleroot) при створенні блоку я використовую власноруч написаний алгоритм (рис. 2.2).

merkle.py > merkle_root

You, a month ago | 1 author (You)

```python
import hashlib

def merkle_root(ts):
    transactions = None
    if isinstance(ts, list):
        transactions = ts.copy()
    else:
        return(hashlib.sha256(ts.encode('utf-8')).hexdigest())

    for i in range(len(transactions)):
        transactions[i] = hashlib.sha256(transactions[i].encode('utf-8')).digest()

    while len(transactions) > 1:
        i = 0
        if len(transactions) % 2 != 0:
            transactions.append(transactions[-1])

        while i < (len(transactions) - 1):
            transactions[i] = hashlib.sha256(transactions[i] + transactions[i + 1]).digest()
            transactions.remove(transactions[i + 1])
            i = i + 1

    return (transactions[0]).hex()
```

You, a month ago • [Refactoring] Reorganized Block class.

Рисунок 2.2 Алгоритмічна реалізація знаходження хеш-дерева Меркла

2.2.4 Майнінг та алгоритм визначення черги майнінгу вузла

Майнінг, також видобування (від англ. mining — видобуток корисних копалин) — діяльність з підтримки розподіленої платформи і створення нових блоків з можливістю отримати винагороду в формі емітованої валюти і комісійних зборів у різних криптовалютах, зокрема в Біткоїні. Вироблені обчислення потрібні для забезпечення захисту від повторного використання одних і тих же одиниць валюти, а зв'язок майнінгу з емісією стимулює людей витрачати свої обчислювальні потужності і підтримувати роботу мереж. Майнінг це один із видів забезпечення довіри між вузлами, що називається proof-of-work (доказ виконання роботи). В нашому випадку систему не потрібно забезпечувати подібним механізмом, адже у нас немає ні емісії, ні вбудованої валюти. Вузол, що створює блок не отримує абсолютно нічого. В приватних блокчейн філософія децентралізації створення блоків побудована насамперед на тому, що кожний вузол може брати безпосередню участь у верифікації транзакцій та верифікації й створенні нових блоків. Таким чином кожний вузл має певний вплив на глобальну децентралізовану систему та може ідентифікувати спробу підробки на кожному з етапів. В нашому випадку кожний учасник логістичного ланцюгу бачить та контролює всі зміни в блокчейні, без згоди кожного з учасників неможливі ніякі глобальні зміни в базі даних. Наприклад, підприємство, що передано перевізнику важливий вантаж зможе слідкувати за всіма подіями, що з ним відбуваються, і вразі чого, дізнається на якому етапі відбулися проблеми.

Кожний вузол в будь-який момент може створити блок, але щоб не відбувалося плутаниці, в приватних блокчейн зазвичай вводять певну чергу, за якою всі вузли створюють блоки. В нашому випадку алгоритм виглядає наступним чином: коли вузол активується він описує всі дружні вузли та запитує у них їх номер в черзі, обираючи для себе найменш можливий (рис. 2.3). Тобто, якщо в мережі зараз три вузли з номерами 1, 3 та 4, то наш вузл присвоїть собі номер 3 в черзі.

```
def find_miner_queue_number(self):
    queue = api_wraper.get_miner_queue_number_list(self.friendly_nodes)
    for num in range(len(queue)):
        if num in queue or str(num) in queue:
            continue
        return num
    return len(queue)
```

Рисунок 2.3 Визначення номеру в черзі поточного вузла

Далі кожний вузл про себе рахує який за номером в черзі вузл зараз виконує майнінг блоку та очікує новий блок протягом певного часу (таймаут). Якщо таймаут закінчується, то кожний вузлінкрементує номер в черзі та знову запускає таймаут. Тайм Аут визначається періодом внутрішнього scheduler-a (планувальник задач). Коли вузл в рамках своєї черги створює новий блок, він відправляє цей блок всім своїм дружнім вузлам, інкрементує внутрішній номер черги та перезапускає задачу scheduler-a. Кожний вузол, що отримує новий блок проводить його валідацію, приймає рішення про його ставлення до свого блокчейну та інкрементує глобальний номер черги майнінгу, при цьому також перезапускаючи задачу scheduler-a. Якщо після інкременту глобальний номер черги збігається з номером черги поточного вузла, то він одразу пробує створити новий блок (схема реалізації в Додатку А). Таким чином при великому навантаженні pendingpool транзакціями вузли будуть швидко створювати нові блоки.Номер черги не інкрементується безкінечно, кожного разу йде опитування активних дружніх вузлів на максимальний номер в черзі. Коли номер перевищено, то номер в черзі стає таким, що дорівнює мінімальному номеру черги активного вузла.Навіть якщо транзакцій для створення нового блоку в pendingpool немає, то номер черги все одно постійно змінюється. Варто помітити, що в приватному блокчейні важливо, щоб кожний вузол знав про існування всіх інших. Адже тільки в такому випадку є можливим побудова адекватної черги майнінгу без колізій.

2.2.5 Генерація публічних та приватних ключів

Приватний ключ в блокчейні це завжди якесь певне велике число, послідовність випадкових байтів. Ми не хочемо генерувати закритий ключ, що належить комусь іншому, тому виникає необхідність в особливому алгоритмі для генерації рандомних послідовностей. Bitcoin використовує еліптичні криві для генерації секретних ключів.

Крива, яка використовується Bitcoin, може випадковим чином вибирати число між 0 і 2^{256} (що становить приблизно 10^{77}, на замітку, атомів в видимої всесвіту десь між 10^{78} і 10^{82}). Така межа означає, що майже неможливо згенерувати один і той же закритий ключ двічі.

Алгоритм генерації такого приватного ключа я реалізував а рамках цієї дипломної, його програмну реалізацію можна подивитися в Додатку Б. З приватного ключа потім отримується публічний ключ за допомогою алгоритму ECDSA (рис. 2.4). ECDSA (EllipticCurve Digital Signature Algorithm) - алгоритм з відкритим ключем для створення цифрового підпису, аналогічний за своєю будовою DSA, але визначений, на відміну від нього, не над кільцем цілих чисел, а в групі точок еліптичної кривої.

Стійкість алгоритму шифрування ґрунтується на задачі дискретного логарифма в групі точок еліптичної кривої. На відміну від задачі простого дискретного логарифма і задачі факторизації цілого числа, не існує суб експоненціального алгоритму для задачі дискретного логарифма в групі точок еліптичної кривої. З цієї причини «сила на один біт ключа» істотно вище в алгоритмі, який використовує еліптичні криві.

Для отримання публічного ключа з приватного та для генерації цифрового підпису я використав бібліотеку ecdsa та еліптичну криву secp256k1 (рис. 2.5).

```python
def privToPub(private_key):
    priv_key_bytes = codecs.decode(private_key, 'hex')
    key = ecdsa.SigningKey.from_string(priv_key_bytes, curve=ecdsa.SECP256k1).verifying_key
    key_bytes = key.to_string()
    key_hex = codecs.encode(key_bytes, 'hex')
    btc_byte = b'04'
    public_key = btc_byte + key_hex
    return public_key.decode('utf-8')
```

Рисунок 2.4 Отримання публічного ключа з приватного

```python
def digital_signature(private_key, message):
    private_key_bytes = codecs.decode(private_key, 'hex')
    sk = ecdsa.SigningKey.from_string(private_key_bytes, curve = ecdsa.SECP256k1)
    signed_msg = sk.sign(message.encode('utf-8'))
    return (signed_msg.hex(), privToPub(private_key))
```

Рисунок 2.5 Генерація цифрового підпису

За допомогою бібліотеки ecdsa можна також верифікувати підписаний хеш транзакції за його публічним ключем (рис. 2.6). Таким чином якщо хоч 1 біт в транзакції буде змінено, то верифікація цифрового підпису буде негативною.

```python
def sign_verify(signed_message, public_key, message):
    vk = ecdsa.VerifyingKey.from_string(bytes.fromhex(public_key[2:]), curve=ecdsa.SECP256k1)
    try:
        vk.verify(bytes.fromhex(signed_message), message.encode('utf-8'))
    except:
        return False
    return True
```

Рисунок 2.6 Верифікація цифрового підпису

До будь-якого блокчейну завжди додається певний user-cli (користувацький консольний інтерфейс). Це вважається найбезпечнішим використанням блокчейну, коли створення нових транзакцій відбувається напряму вузли блокчейну без

посередників у вигляді веб-інтерфейсів. За допомогою user-cli можна згенерувати собі новий приватний ключ, зашифрувати його ключ в WIF-формат (рис. 2.9), створити та пошири транзакцію. Формат імпорту гаманця (WIF, також відомий як формат експорту Wallet) - це спосіб кодування приватного ключа ECDSA, щоб полегшити його копіювання. В нашому блокчейні приватний ключ вузла зберігається тільки в WIF-форматі.

```
def privToWif(private_key):
    versioned_key = "80" + private_key
    priv_hash_1 = hashlib.sha256(binascii.unhexlify(versioned_key)).hexdigest()
    priv_hash_2 = hashlib.sha256(binascii.unhexlify(priv_hash_1)).hexdigest()
    binary_data = versioned_key + str(priv_hash_2)[:8]
    wif = base58.b58encode(binascii.unhexlify(binary_data))
    return wif.decode('utf-8')
```

Рисунок 2.7 Перетворення приватного ключа в WIF-формат

2.2.6 Серіалізація даних

Серіалізація — процес перетворення будь-якої структури даних у послідовність бітів. Зворотною до операції серіалізації є операція десеріалізації — відновлення початкового стану структури даних із бітової послідовності. Для зручності зберігання даних та передачі транзакцій і блоків в мережі я реалізував алгоритм серіалізації (рис. 2.8) та десеріалізації (рис. 2.9) транзакцій.

Транзакції в блоках, або в pendingpool зберігаються тільки в серіалізованому вигляді. Оскільки інформація про вантаж зберігається тільки в першій транзакції, то в решті транзакцій інформація передається у вигляді строки «x80». Це зроблено для зручності серіалізації та десеріалізації транзакції.

```
class Serializer:
    @staticmethod
    def serialize(tx):
        information_serialized = ""

        if tx.information != NAN:
            information_serialized = "0" * (INFROMATION_LEN - len(tx.information)) + tx.information
        else:
            information_serialized = tx.information

        return ('0' * (CARGO_ID_LEN - len(tx.cargo_id)) + tx.cargo_id +
                ('0' * (TIMESTAMP_LEN - len(tx.timestamp))) + tx.timestamp +
                ('0' * (TEMPERATURE_LEN - len(tx.temperature)) + tx.temperature) +
                information_serialized +
                tx.public_key +
                tx.signed_hash)
```

Рисунок 2.8 Алгоритм серіалізації транзакції

```
class Deserializer:
    @staticmethod
    def deserialize(serialized):
        information_length = 0
        if serialized[CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN: CARGO_ID_LEN +
                TIMESTAMP_LEN + TEMPERATURE_LEN + len(NAN)] == NAN:
            information_length = len(NAN)
        else:
            information_length = INFROMATION_LEN
        d = { "cargo_id": serialized[:CARGO_ID_LEN].lstrip("0"),
                "timestamp": serialized[CARGO_ID_LEN:(CARGO_ID_LEN + TIMESTAMP_LEN)].lstrip("0"),
                "temperature" : serialized[CARGO_ID_LEN + TIMESTAMP_LEN :
                        CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN].lstrip("0"),
                "information" : serialized[CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN :
                        CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN + information_length].lstrip("0"),
                "public_key": serialized[(CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN + information_length):
                        (CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN + information_length + PUBLIC_KEY_LEN)],
                "signed_hash": serialized[(CARGO_ID_LEN + TIMESTAMP_LEN + TEMPERATURE_LEN + information_length + PUBLIC_KEY_LEN):] }
        return Transaction.from_dict(d)
```

Рисунок 2.9 Алгоритм десеріалізації транзакції

2.2.7 Валідація транзакцій

Кожна транзакція, що потрапляє до вузла через API, проходить верифікацію перед тим як потрапити до pendingpool. Це зроблено для того, щоб заздалегідь некоректні транзакції не займали зайве місце. Також, при створені блоку, вузол

забирає транзакцію з пулу та знову проводить верифікацію. Транзакція могла бути сфабрикована на етапі збереження, тому потрібно засвідчити, що це не так.

В процесі верифікації транзакції йде перевірка цифрового підпису та наявності прав у власника приватного ключа на створення нових транзакцій (рис. 2.10). Кожний вузол зберігає публічні ключі всіх, хто може створювати нові транзакції.

Вони додаються в спеціальний файл на етапі ініціалізації блокчейну. Процес видалення або добавлення нових публічних ключів до цього списку можна реалізувати за допомогою смарт-контракту, що потребує підпису кожного вузла.

```
def validate_transaction(t):
    if t is None:
        return ReturnCode.INVALID_TRANSACTION
    if check_transactions_permissions(t) is False:
        return ReturnCode.UNAUTHORIZED_PRIVATE_KEY
    if check_digital_signature(t) is False:
        return ReturnCode.INVALID_SIGNATURE
    return ReturnCode.OK

def check_digital_signature(t):
    return wallet.sign_verify(t.signed_hash, t.public_key, t.calculate_hash())

def check_transactions_permissions(t):
    return t.public_key in FileSystem.getPermissionedCheckpointsPublicAddresses()
```

Рисунок 2.10 Процес верифікації транзакції

2.2.8 Валідація блоків та консенсус

Кожний блок, що надходить до вузла проходить валідацію. В процесі верифікації блоку перевіряється: наявність прав на створення блоків, коректність хешу блоку та його цифрового підпису, коректність Merkleroot (Рис. 2. 11).

```
def validate(block):
    if block is None:
        return ReturnCode.INVALID_BLOCK
    if check_block_creation_permissions(block) is False:
        return ReturnCode.UNAUTHORIZED_PRIVATE_KEY
    if check_merkle_root(block) is False:
        return ReturnCode.INVALID_MERKLE_ROOT
    if check_digital_signature(block) is False:
        return ReturnCode.INVALID_SIGNATURE
    if check_block_hash(block) is False:
        return ReturnCode.INVALID_BLOCK_HASH
    return ReturnCode.OK

def check_merkle_root(block):
    return merkle.merkle_root(block.transactions) == block.merkle_root

def check_block_hash(block):
    return block.hash == block.calculate_hash()

def check_digital_signature(block):
    return wallet.sign_verify(block.signed_hash, block.public_key, block.calculate_hash())

def check_block_creation_permissions(t):
    return t.public_key in FileSystem.getPermissionedValidatorsPublicAddresses()
```

Рисунок 2.11 Процес верифікації блоку

Під час валідації Merkle root проходять валідацію всі транзакції, що містяться в блоці. Це допомагає значно зменшити витрати на верифікацію блоку та позбавляє необхідності десереалізовувати кожну транзакцію.

Консенсус в загальному розумінні означає спосіб прийти до угоди. У публічному блокчейні, який являє собою децентралізовану систему, яка не має єдиного керуючого органу, для досягнення консенсусу розроблені різні алгоритми. При розробці блокчейну завжди являє собою велику проблему реалізація якісного алгоритму консенсусу. В нашому випадку ми маємо приватний блокчейн. Алгоритм консенсусу як такий не потрібен. Вузл приймає всі блоки, які проходять верифікацію. Щоб сфабрикувати транзакцію, потрібно мати безпосередній доступ до авторизованого приватного ключа контрольної точки. Такі транзакції будуть одразу замічені в системі, та можна прийняти рішення про видалення цього приватного ключа зі списку авторизованих. Фабрикуючи транзакції неможливо завдати великої шкоди для блокчейну, адже вони нічого не змінюють, а просто добавляють нову інформацію, про недостовірність якої можна сповіщати користувача через відповідний інтерфейс.

Прикладний програмний інтерфейс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) (англ. Application Programming Interface, API) — набір визначень підпрограм, протоколів взаємодії та засобів для створення

програмного забезпечення. API в блокчейні дозволяє будувати різного роду користувацькі інтерфейси без прив'язки до певних правил реалізації. Тобто будь-який розробник, при бажанні може написати свій сайт для дослідження блокчейну. Кожна сторона нашого ланцюгу поставок може розробити свій інтерфейс взаємодії з блокчейном та позбутися необхідності в довірі стороннім сервісам. Це одне із основних правил в філософії побудови блокчейну.

В рамках цієї дипломної роботи за допомогою API розробленого блокчейну відбувається поширення нових транзакцій та блоків по мережі, встановлення черги майнінгу, дослідження стану транзакцій, блоків та pendingpool. Звертаючись по посиланню будь-якого вузла ви по замовчуванню отримаєте HTML сторінку, де повністю описаний API.

Кожний запит в рамках розробленого API буде повертати JSON-об'єкт, що складається з коду повернення (англ. returncode) та даних. Якщо запит був невдалим, то ви отримаєте об'єкт з кодом помилки та пустими даними. Програмну реалізацію API, опис кожного роута та всіх можливих кодів повернення ви можете подивитись в Додатку. Тут я опишу один із найважливіших роутів: /transaction/new – роут, за допомогою якого можна поширювати нову транзакцію. Виконуючи post-запит з використанням цього роуту слід передати JSON-об'єкт з серіалізованою підписаною транзакцією, або у вигляді словника (англ. dictionary) (рис. 2.14). При цьому слід вказати що саме ви передаєте: dictionary або serialized.

```
url = 'http://127.0.0.1:5100/transactions/new'

t = Transaction("134589037", '1588410854.301351', "25", "cargo_info")

private_key_wif = "5HpXfPm94cFqqZnQwZ5QnLrb4adRU9CrJXtxYWETxcnpWAPir61"
private_key = wallet.wifToPriv(private_key_wif)

t.sign(private_key)

obj = {'dictionary' : t.to_dictionary()}

x = requests.post(url, json = obj)
print(json.loads(x.content.decode('utf-8')))
```

Рисунок 2.12 Post-запит з новою транзакцією в форматі dictionary

Весь процес створення транзакції та її підпис приватним ключем користувача повинен відбуватися на стороні користувача. Приватний ключ не повинен нікуди відправлятися. Якщо приватний ключ буде сфабриковано, то зловмисник може створювати нові транзакції від імені власника. Математично визначити приватний ключ знаючи публічний ключ та цифровий підпис практично неможливо, тому потрібно просто приділити достатньо уваги збереженню конфіденційності вашого приватного ключа. В процесі тестового post-запиту я використав приватний ключ в wif-форматі, публічний ключ якого авторизований для створення нових транзакцій. Якщо код повернення дорівнює нулю, то ваш запит пройшов успішно і транзакція була додана до memorypool.

2.3 Проектування та розробка зв’язку радіомітки з блокчейном

Фізична сторона розробленоїзвсистеми по контролю доставки вантажів явля’єтьсярадіомітка, що кріпиться на вантаж та радіо читач, за допомогою якого можна зчитати інформацію з радіомітки. Радіомітка повинна мати в собі унікальний ідентифікатор, що не можна переписати. Таким чином, цей індетифікатор або буде являти собою безпосередньо ідентифікатор вантажу та буде встановлювати зв’язок реального вантажу з його цифровою інформацією в блокчейні, або буде виконувати роль верифікатора автентичності даних.

2.3.1 Аналіз існуючих радіоміток

Радіочастотне розпізнавання здійснюється за допомогою закріплених за об'єктом спеціальних міток, що несуть ідентифікаційну та іншу інформацію. Цей метод вже став основою побудови сучасних безконтактних інформаційних систем, і має стійку назву RFID-технології.

Особливості технології:

- RFID-міткам не потрібен контакт або пряма видимість, дані з мітки можуть бути отримані на відстані.

- RFID-мітки зчитуються швидко і точно, що дозволяє виконувати велику кількість сканувань.

- RFID-мітки можна використовувати навіть в агресивних середовищах, через бруд, фарбу, пар, воду, пластмасу, деревину, а також використовувати імплантацію в тіло.

- Пасивні RFID-мітки мають фактично необмежений термін експлуатації, мають низьку собівартість.

- RFID-мітки можуть нести велику кількість інформації.

- RFID-мітки легко відстежити на порівняно невеликій відстані — метро, офіси, банки, магазини, зупинки.

- RFID-мітки можуть бути використані як для читання, так і для запису великого обсягу інформації.

RFID мітки класифікують по дальності зчитування, наявності джерела живлення, типу пам'яті, робочій частоті і деяким іншим параметрам. За дальності розрізняють мітки ближньої дії (до 20 см), середньої (від 0,2 до 5 м) і високої дальності (від 5 до 100 м). За типом споживання мітки поділяють на пасивні (харчування відбувається через наведений в антені сигнал зчитувача) з радіусом дії сигналу не більше 10 м, активні з власним джерелом живлення і радіусом зчитувача до 300 м, і напівпасивні (або напівактивні), також мають власне джерело живлення. По виду пам'яті розрізняють такі види RFID-міток:

- RO (одноразового запису)

- WORM (з блоком одного разу записаної пам'яті для багаторазового зчитування)

- RW (з вбудованим блоком пам'яті для багаторазового запису та зчитування даних)

За частотою діапазону розрізняють наступні типи RFID-міток:

- Низькочастотні (LF) (125-134 кГц). Використовуються в системах контролю і управління доступом, домофонах, чіпування тварин.

- Високочастотні (HF) (13,56 МГц).

- Ультра високочастотні (UHF) (860-960 МГц). Використовуються для маркування різного роду фізичного майна (логістика, склади).

Також За останні роки набрала популярності технологія NFC. Ця технологія набагато молодша версія RFID. Радіус її дії становить максимум 10 см і в ній може бути встановлений як односторонній, так і двосторонній зв'язок. NFC означає комунікація ближнього радіусу дії, RFID - радіочастотна ідентифікація. Обидві технології використовують радіосигнал для пошуку міток і відстеження цілей, приходять на зміну штрих кодування. NFC тільки зароджується, RFID вже широко поширена в усьому світі. Давайте розглянемо одностороннюю передачу даних: використовуючи смартфон, ви можете прочитати NFC тег, який може бути вбудований в рекламні постери, політичні листівки, путівники або в куплений товар. Розумні мітки дуже схожі на RFID теги, вони просто налаштовані на роботу з NFC-зчитувачами, замість RFID. NFC технології являються більш комплексними. Уже станом на 2014 рік в 50% смартфонів був вбудований NFC чіп, що перетворив телефон в цифровий гаманець.

Наразі ця цифра вже значно більша. Торкаючись телефоном до NFC терміналу, NFC чіп автоматично переходить в режим емуляції карти. Одним махом можна оплатити товари в супермаркеті. Це так звана безконтактна форма оплати. Ваш телефон зараз може замінити кредитні, бонусні, транспортні карти, здійснюючи оплату набагато швидше і зручніше.

2.3.2 Вибір радіомітки для дипломної роботи

В реальному проекті слід використовувати мітки, що підтримують криптографічно захищену аутентифікацію читача, мають механізм для вимірювання критичні показники температури перевезення. Але для простоти реалізації прототипу в рамках цієї дипломної роботи я буду використовувати високочастотну радіомітку NTAG213 ISO 14443A (рис. 2.13). Це клейка мітка зі спеціальним чіпом, що забезпечує ближню безконтактний зв'язок NFC. Таким чином кінцевий

користувач зможе зчитати ідентифікатор отриманого вантажу телефоном та за допомогою користувацького інтерфейсу блокчейну отримати всю наявну інформацію про вантаж.

Рисунок 2.13 NTAG213 ISO 14443A

Технічні характеристики радіомітки NTAG213 ISO 14443A:

- Доступна пам'ять: 144 байт
- Стандарт: ISO 14443A
- Робоча частота: 13.56 МГц
- Дистанція зчитування: до 5 см
- Кількість циклів запису / зчитування: 100000
- Діаметр наклейки: 2.5 см

Для роботи з чіпами потрібен спеціальний зчитувач або гаджет з підтримкою NFC. Для запису інформації на чіп з телефону потрібно мати спеціальне програмне забезпечення, яке можна завантажити з відповідного маркету (пошук за словом NFC).

2.3.3 Проектування радіо читача

Радіо Читач повинен вміти не тільки зчитувати/записувати інформацію, а й формувати транзакцію, підписувати її авторизованим приватним ключем та відправляти до блокчейну. Тому потрібне комплексне рішення. В реальному проекті читачі слід прошивати зашифрованою прошивкою на етапі ініціалізації блокчейну. Приватний ключ повинен зберігатися в пам’яті програми в зашифрованому вигляді

для запобігання зловмисного reverse engineering. В рамках цієї дипломної роботи я буду використовувати RaspberryPi 4 Model B та читач RFID-RC522.

Raspberry Pi — одноплатний комп'ютер, розроблений британским фондом Raspberry Pi Foundation. RaspberryPi побудований на системі-на-чипі (SoC) Broadcom BCM2835, яка включає в себе процесор ARM із тактовою частотою 700 МГц, графічний процесор VideoCore IV, і 512 чи 256 мегабайтів[4] оперативної пам'яті. Твердий диск відсутній, натомість використовується SD карта. Мною було встановлену операційну Raspbian (Debian для RaspberryPi).

Радіочитач - це модуль виконаний на мікросхемі MFRC522 фірми NXP, яка забезпечує роботу з мітками HF (на частоті 13,56 МГц) (рис. 2. 14).

Технічні характеристики RFID-модуля RC522:

- Напруга живлення: 3.3V
- Струм: 13-26mA
- Робоча частота: 13.56MHz
- Дальність зчитування: 0 - 60 мм
- Інтерфейс: SPI
- Швидкість передачі: максимальна 10Мбіт / с
- Розмір: 40мм х 60мм

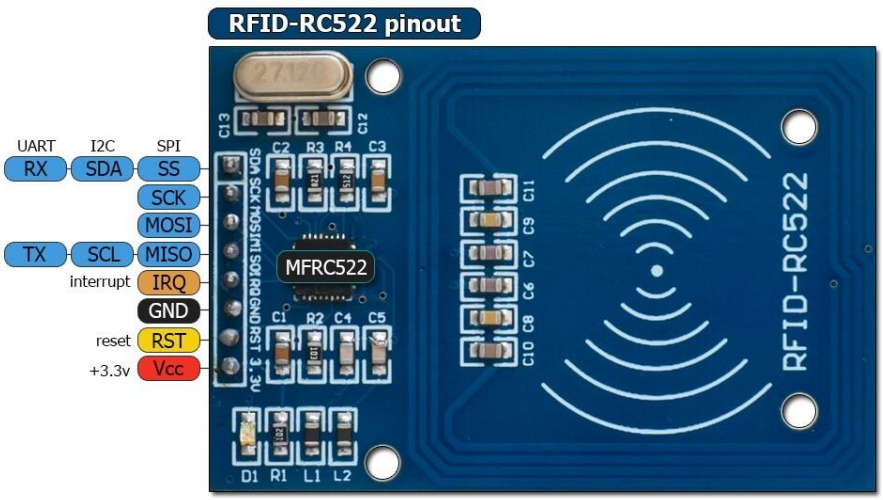


Рисунок 2.14 Модуль RFID-RC522

Мікросхема MFRC522 підтримує інтерфейси SPI, UART і I2C. Вибір інтерфейсу здійснюється установкою логічних рівнів на певних висновках мікросхеми. На даному модулі обраний інтерфейс SPI. SPI (англ. Serial Peripheral Interface) — фактичний послідовний синхронний повнодуплексний стандарт передачі даних, розроблений фірмою Motorola для забезпечення простого сполучення мікроконтролерів та периферії.

По інтерфейсу SPI я підключив MFRC522 до Raspberry PI (рис. 2.15).

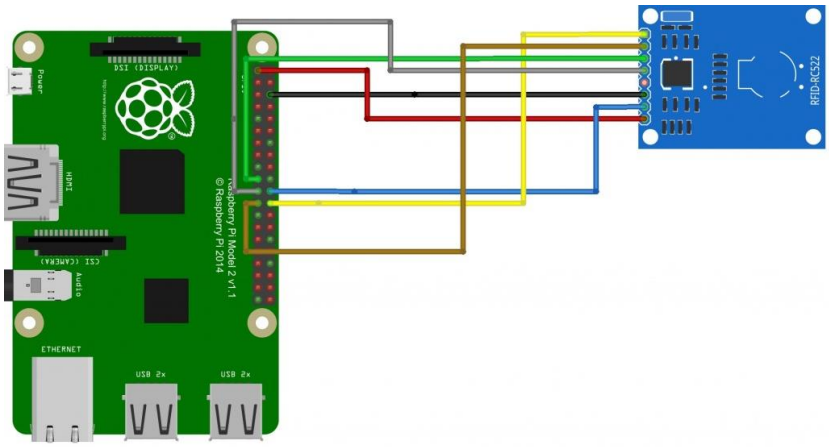


Рисунок 2.15 Схема підключення MFRC522 до Raspberry PI

Так само як з приватним ключем, посилання на вузол, куди читач має надсилати свої транзакції, має бути вшите в програмний код читача на етапі ініціалізації системи для логістичного ланцюга. Зручно вказувати той вузл, власником якого являється контрольна точка на якому буде використовуватися читач. Для відправлення та створення транзакції була розроблена рекурсивна програма на мові програмування Python, що кожної секунди намагається зчитати інформацію з радіо читача та при успішному зчитуванні вимірює температуру та відправляє транзакцію (рис. 2. 16).

```
url = 'http://192.168.0.101:5101/transactions/new'
private_key_wif = "5HpXfPm94cFqqZnQwZ5QnLrb4adRU9CrJXtxYWETxcnpWAPir61"
private_key = wifToPriv(private_key_wif)
reader = Reader()
thermometer = Thermometer()

def send_cargo_id(cargo_id):
    t = Transaction(str(cargo_id), str(time.time()), thermometer.get_temparature())
    t.sign(private_key)
    data = requests.post(url, json = {'dictionary': t.to_dictionary()})
    print(data.content)

def reader_periodic():
    data = reader.read()
    if data is not None:
        print(data)
        time.sleep(1)
        send_cargo_id(data[0])
    reader_periodic()

reader_periodic()
```

Рисунок 2.16 Рекурсивна реалізація читача

2.4 Проектування та розробка користувацького інтерфейсу

Для дослідження блокчейну або створення нових транзакцій власноруч потрібен якийсь певний користувацький інтерфейс. Розроблений API блокчейну дозволяє кожному бажаючому створити свій інтерфейс та реалізувати на них будь-яку логіку. Наприклад, періодичні патерни пошуку та повідомлень про якийсь товар. Також це дає впевненість, що приватний ключ ніким не буде сфабриковано. Користувацький інтерфейс це всього лиш інтерфейс і він не є головною ідеєю дипломної роботи, проте є його важливою частиною.

В рамках цієї дипломної роботи було розроблено SPA (single page application) за допомогою мови програмування JavaScript, фреймворка React та мов розмітки HTML та CSS. Його вигляд можна побачити в 3 розділі.

```
import * as axios from "axios";

const instance = axios.create({
    baseURL: 'http://192.168.0.109:5101/',
    headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json'
    }
})

export const itemsAPI = {
    getItems (cargoId) {
        console.log(cargoId)
        return instance.get('/find/?cargo_id=' + cargoId)
    },
    postItems(dictionary) {
        console.log(dictionary)
        return instance.post(`/transactions/new`, dictionary)
    }
}
```

Рисунок 2.17 Спосіб використання API через ReactJS

Таким чином, використовуючи розроблений API можна побудувати веб-сайт будь-якого формату.

Висновки за розділом 2

Ми ознайомилися з архітектурою проекту і знайшли найбільш ефективне рішення створення цієї програми.

Ми розібрали такі пункти:

- Модуль RFID міток
- Схема підключення до Raspberry Pi
- Рекурсивна реалізація читача
- Post запит до API
- Верифікація блоку
- Як підписати приватний ключ
- WIF формат ключа
- Як працює публічний ключ

РОЗДІЛ 3
ОПИС РОБОТИ СИСТЕМИ З БОКУ КОРИСТУВАЧА

3.1 Процес Розгортання блокчейну для логістичного ланцюга

Розроблена система розгортається безпосередньо на логістичний ланцюг підприємства або організації.

На початку розгортання блокчейну власником генерується потрібна кількість приватних ключів. Вони розподіляються між вузлами та контрольними точками. Потім Ініціалізується потрібна кількість вузлів. Кожна бажаюча сторона може ініціалізувати свій вузол та, маючи авторизований приватний ключ, брати участь у валідації та створенні нових блоків.

3.2 Процес створення нових транзакцій на контрольних точках

На всіх контрольних точках, крім початкової, створення та поширення транзакції відбувається безпосередньо через програму радіо читача. Таким чином можна пропускати велику кількість вантажів через конвеєр з радіо зчитувачем, при цьому формуючи транзакції та відправляючи їх до блокчейну. На першій контрольній точці, що являє собою певну організацію або підприємство, важливо вказати унікальну інформацію про вантаж, яку потім неможливо буде змінити. Це можна також реалізувати через зв'язування бази даних підприємства з читачем, який буде поширювати транзакції, але для наглядності я реалізував це за допомогою користувацького інтерфейсу в розробленому веб-сайті. Для створення такої транзакції потрібно вказати ідентифікатор вантажу, всю наявну інформацію про нього та вказати авторизований приватний ключ в WIF-форматі для підписування транзакції (рис. 3. 1).

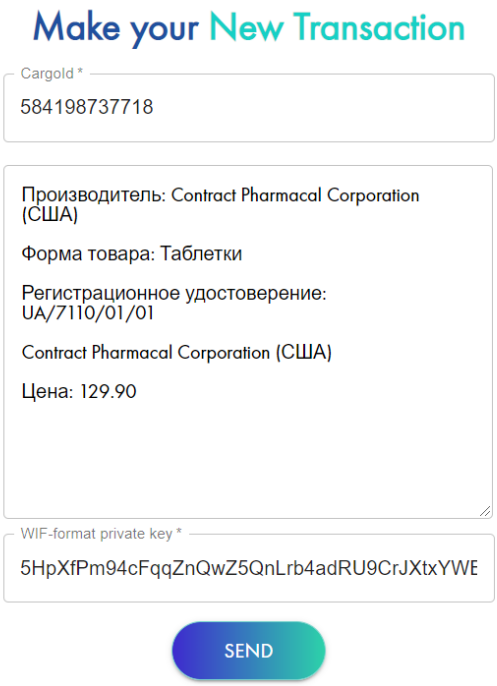


Рисунок 3.1 Створення першої транзакції про вантаж

3.2.1 Відстежування стану вантажу з боку кінцевого користувача

Розроблений API блокчейну дозволяє робити його дослідження в будь-який момент. Можна отримати всі існуючі блоки або блок будь-якої висоти для перевірки коректності цифрових підписів, або для отримання певної інформації. Блокчейн повністю прозора та відкрита база даних. Для кінцевого користувача важливо знати про автентичність його товару та умови перевезення. Це можна зробити за допомогою користувацького інтерфейсу. Спочатку необхідно дізнатися та ввести ідентифікатор вантажу (рис. 3. 2)

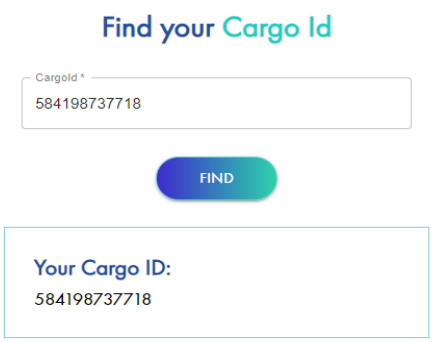


Рисунок 3.2

Після цього вам буде вся відома інформація про вантаж в зручному для користувача вигляді (рис. 3.3)

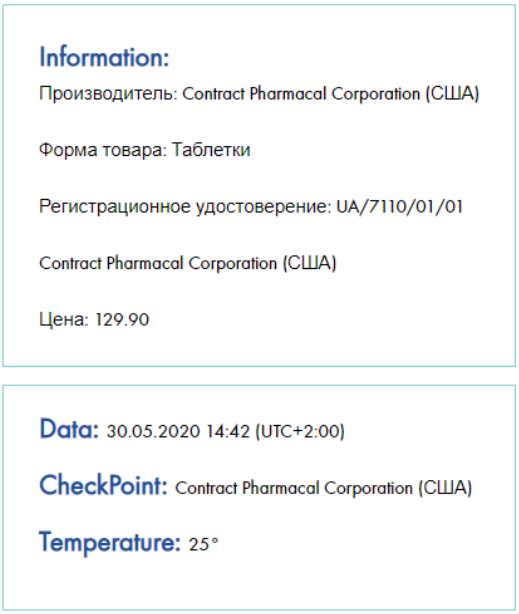


Рисунок 3.3 Інформація про обраний вантаж

Таким чином користувач може дізнатися достовірну інформацію про його вантаж на кожному етапі доставки.

3.2.2 Розбір роботи Frontend архітектури та системи

React-розробка полягає в описі того, що потрібно вивести на сторінку (а не в складанні інструкцій для браузера, присвячених тому, як це робити). Це, крім іншого, означає значне скорочення обсягів шаблонного коду.

У складі Angular, з іншого боку, є засоби командного рядка, які генерують шаблонний код компонентів. Чи не здається це трохи не тим, чого можна чекати від сучасних інструментів розробки інтерфейсів? Фактично, мова йде про те, що в Angular так багато шаблонного коду, що для того, щоб його генерувати, навіть створено спеціальний засіб.

У React, приступаючи до розробки, просто починають писати код. Тут немає шаблонного коду компонентів, який потрібно якось генерувати. Звичайно, перед розробкою потрібна деяка підготовка, але, коли справа доходить до компонентів, їх можна описувати у вигляді чистих функцій.

README.md App.js

```
import React from 'react';
import './App.css';
import {Route, Switch} from "react-router-dom";
import {Main} from "./Main";
import {Find} from "./components/FindCargo/Find/Find";
import {NewTransaction} from "./components/NewTransaction/NewTransaction/NewTransaction";
import {Pendings} from './components/Pendings/Pendings/Pendings'


const App = (props) => {
        return (
            <div className='app-wrapper'>
                <Switch>
                    <Route exact={true} path='/' render={() => <Main/>}/>
                    <Route exact={true} path='/find' render={() => <Find/>}/>
                    <Route exact={true} path='/new' render={() => <NewTransaction/>}/>
                    <Route exact={true} path='/pendings' render={() => <Pendings/>}/>
                </Switch>
            </div>
        )
}

export default App;
```

Рисунок 3.4 Як виглядяє наша основна сторінка

Тут (рис. 3.4) ми можемо побачити що використовуємо компонентний стиль написання коду, для цього ми розподіляємо кожен компонент на свої ролі.

<Swtich> відповідає за розподіл Payса між усіма компонентами в нашому веб додатку. Так як ми використовуємо SPA (Single Page Application) все роути зберігаються один раз і нам не потрібно перезавантажувати сторінку кожен раз коли переходимо за новими роутами які ми ввели командою <Route>

Також ми експортуємо компонент App так як вона є основним компонентом всього нашого веб додатки, щоб браузер зміг його скомпілювати з усіма іншими модулями підключеними ми. Як ми знаємо браузер працює з деревом DOM і для цього йому потрібні лише JS об'єкти. React якраз переводить всі компоненти в чисті JS об'єкти з якими браузер легко працює.

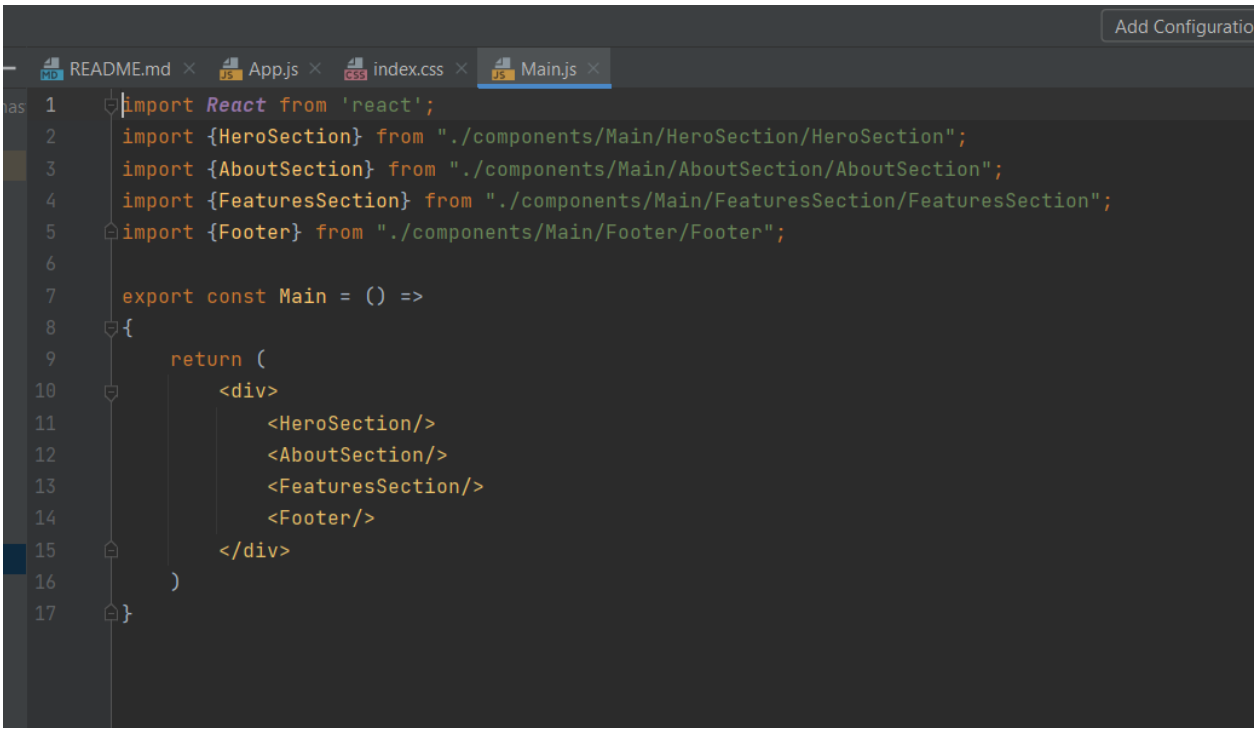


Рисунок 3.5 Як виглядає наша сторінка з сторони браузера

Компоненти в React можна створювати, застосовуючи два підходи. Перший полягає у використанні класів компонентів (Class Component), другий - у використанні функціональних компонентів (Functional Component). Як ви, можливо, помітили, у вищенаведеному прикладі використовуються класи. На жаль, більшість посібників з React для початківців пропонують використовувати саме їх.

Що поганого в описі компонентів з використанням механізму класів? Справа в тому, що такі компоненти важко тестувати і вони мають властивість надмірно розростатися. Ці компоненти схильні до проблеми неякісного поділу відповідальності, змішування логіки і візуального представлення (а це ускладнює

налагодження і тестування додатків). В цілому, використання класів компонентів веде до того, що програміст, фігурально висловлюючись, «стріляє собі в ногу». Тому, особливо якщо мова йде про початківців програмістів, я порекомендував би їм зовсім не користуватися класами компонентів.

3.2.3 Юзер інтерфейс (UI)

Суть React полягає в тому, що його архітектура базується на компонентах. Як в лего, все збирається з цеглинок.

Далі компонент самого верхнього рівня рендерится в певний елемент на сторінці. Різні компоненти можна отрендеріть в різні елементи, ніхто цього не забороняє.

Так само треба чітко розуміти, що філософія React - генерувати розмітку динамічно при зміні Стейт (стану). Класично стейт зберігається локально в кожному компоненті, однак найчастіше це незручно, тому придумали Flux, однією з інкарнацій якого є, в деякому наближенні, Redux - якесь централізоване сховище Стейт, з пляшками і балеринами.

Наскільки мені відомо, React не нав'язує ніяких парадигм і прекрасно спільно уживається на одній сторінці з чим завгодно, так-що, в принципі, отреактіть можна тільки найнеобхідніше, а решту запив по-старому.

UX-дизайнер створює макети призначеного для користувача інтерфейсу, а React-розробники реалізують їх як веб-компоненти, які можуть відправляти і отримувати дані з бекенда-сервера. При цьому важливо розробляти компоненти, які можна використовувати повторно, щоб скоротити час на подальшу розробку.

Приклад завдання, яку може вирішити React-розробник.

Створити кнопку, яка покаже спиннер всередині неї при відправці HTTP-запиту і перетвориться назад в звичайну кнопку при успішному або помилковому виконанні запиту

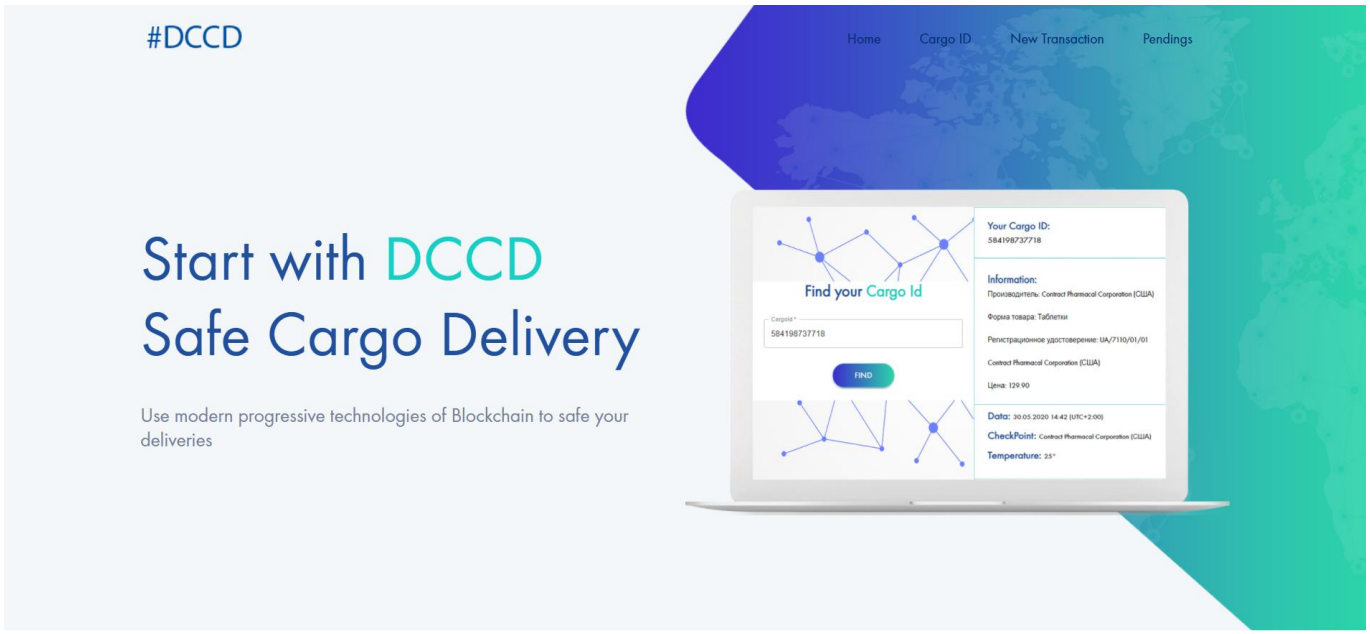


Рисунок 3.6 Перша сторінка нашого веб-интерфейсу

Рисунок 3.7 Друга сторінка нашого веб-интерфейсу

Рисунок 3.8 Футер веб-интерфейсу DCCD

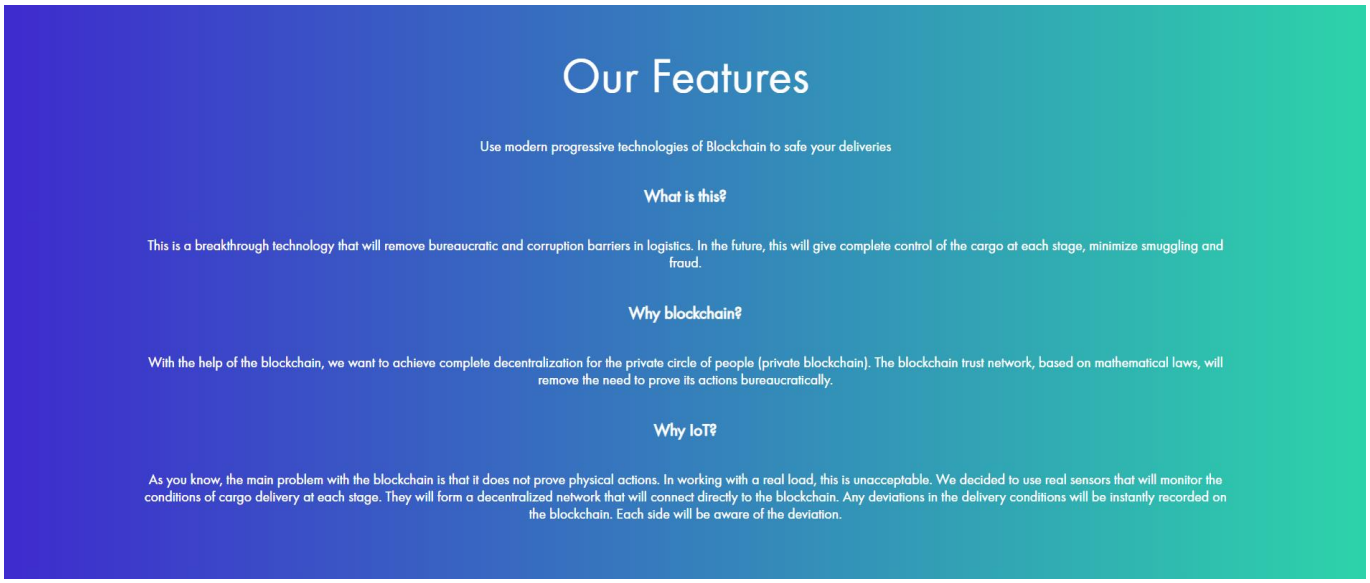


Рисунок 3.9 Наши нови технології у веб-интерфейсу

Наши переваги для кліента:

- Використовуйте сучасні прогресивні технології Blockchain, щоб захистити свої доставки
- Це проривна технологія, яка усуне бюрократичні та корупційні бар'єри в логістиці. У майбутньому це дасть повний контроль над вантажем на кожному етапі, мінімізує контрабанду та шахрайство.
- Чому блокчейн?
- За допомогою блокчейну ми хочемо досягти повної децентралізації для приватного кола людей (приватний блокчейн). Довірча мережа блокчейнів, заснована на математичних законах, зніме необхідність доводити свої дії бюрократично.
- Як відомо, основною проблемою блокчейну є те, що він не доводить фізичних дій. У роботі з реальним навантаженням це неприпустимо. Ми вирішили використовувати справжні датчики, які контролюватимуть умови доставки вантажу на кожному етапі. Вони сформують децентралізовану мережу, яка буде підключатися безпосередньо до блокчейну. Будь-які відхилення в умовах доставки миттєво реєструються на блокчейні. Кожна сторона буде усвідомлювати відхилення.

3.2.4 Github для контролю якості коду

Для запуску нашого коду та збірки нашої програми було прийнято рішення використовувати технологію yarn. За допомогою нього будь-яка людина хто скачав наш код і архітектуру зможе запросто підняти збірку і включити сайт (див. рис.3.10).

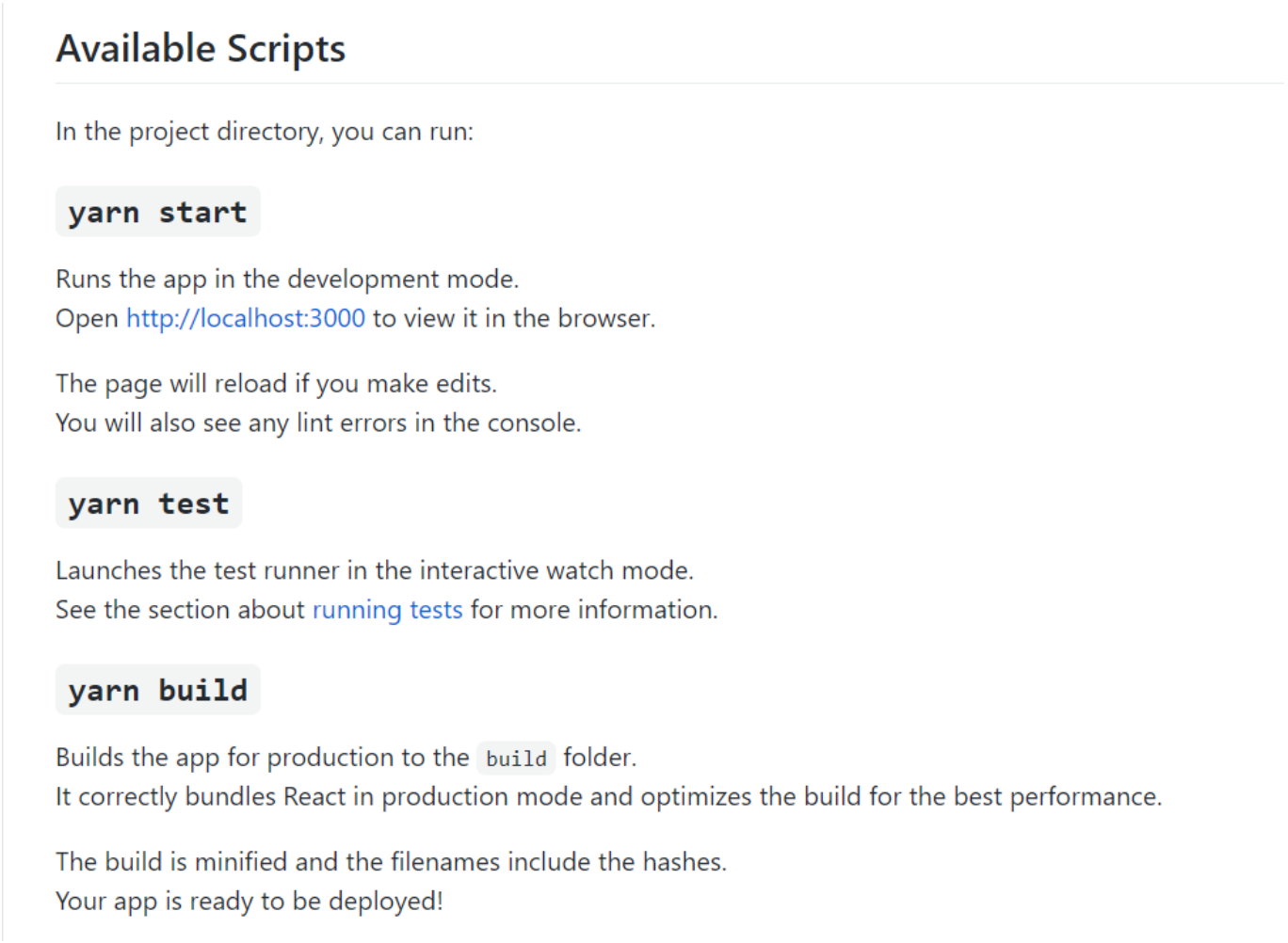


Рисунок 3.10 Сборка yarn

Також github зберігає всі кеші і дані нашого застосування щоб в майбутньому можна було б його набагато швидше зібрати, для цього використовується технологія кешування браузера і стилів. Зберігає він стилі в форматі .css і .html (див. рис.3.11)

22 lines (22 sloc) | 2.31 KB

```
favicon.ico,1588364840468,d96ddbc4933b04e12c738ab39f469573143949ca2c39eda0a49d16f83d40c319
logo192.png,1588364840468,3ee59515172ee198f3be375979df15ac5345183e656720a381b8872b2a39dc8b
logo512.png,1588364840468,ee7e2f3fdb8209c4b6fd7bef6ba50d1b9dba30a25bb5c3126df057e1cb6f5331
manifest.json,1588364840468,aff3449bdc238776f5d6d967f19ec491b36aed5fb7f23ccff6500736fd58494a
robots.txt,1588364840468,bfe106a3fb878dc83461c86818bf74fc1bdc7f28538ba613cd3e775516ce8b49
asset-manifest.json,1591227248503,b3e10d352656d4c60f2d4deb364eca9b68c401b62a3dc87527af0bdb09134f16
index.html,1591227248492,dcc7158fa9366d250d347798e9f9fdbb4da5c59bce4787fe20cbfffa8a1252b6
precache-manifest.a23fb3a0fe7594fda8833e28981240fe.js,1591227248492,5fad234890386f9ab9055e30d14a5c6f8bc32f096cf8024840e1c038ad33efff
service-worker.js,1591227248501,6d557a7b006d8be2bb5efa7ffb7c8709416ad7a1d5325678aa140a16d45420c4
static/css/main.5aa64617.chunk.css,1591227248495,77ecfaab32a619b0ec3e15a1a9a58f6ef5d864a83cd36728693f041ea28b08f2
static/js/2.96a968aa.chunk.js.LICENSE.txt,1591227248503,4b3494bb6d975825e79581eb12dbb0eee21712cfd2cacd15df8b31057188c231
static/css/main.5aa64617.chunk.css.map,1591227248503,67744b36087cea05e7499402f5ebd0a4d53ed5c9e52801d0c7d1cc24455acd8f
static/js/runtime-main.7c0ae09a.js,1591227248501,d008a9456ec370f89f39db3730b71e07517a411425bceb4f06c3f8b98b5d38aa
static/js/main.8fb3064d.chunk.js,1591227248503,dbccd3685256a2150dd06940cf1e4f936271a8cf5a7a71b60b1cb23b7a7d3b47
static/js/runtime-main.7c0ae09a.js.map,1591227248503,51a03ba250f7c5b112071693e34fdcc0f714aff224703abf59808e8f77769685
static/js/main.8fb3064d.chunk.js.map,1591227248503,012bb0ec7c6c48046701f20cabc21d4cf8da0c0609e654e170dc2c86929cb798
static/media/Futura.ee04818d.woff,1591227248492,9469a923ad5977bcbab01ec68342e24fb8ac2918f42ab69abf6fdb378a1ed83b
static/media/hero-bg.96974547.png,1591227248503,ba48dfe1bdae1c489f1c6ea8c7c2cc2106c28ae4870cb2e29225fa25577147c3
static/media/laptop.70efb696.png,1591227248503,366745db526a460ef0727797df5ffcf1c19bd9d87698f92add09f1cd20c2e407
static/js/2.96a968aa.chunk.js,1591227248503,9c4b83b58a6204503220a2ab86ace9406bb56f026aafef5eb8a062b8e6227d4f
static/media/about-img.bb4d9c0c.bmp,1591227248504,42158c1fa276d3011c5de12ef28682a8723829b5a0f248c14cef1a51b11bcec6
static/js/2.96a968aa.chunk.js.map,1591227248506,3b6cf4e3bf928fecbc24f61d69be9a54a309d91dd7a1cf1f411853e7e4b06f65
```

Рисунок 3.11 Кешування даних

3.3 Зв'язка React і Redux

Що таке Redux?

Redux - це передбачуваний контейнер стану, розроблений, щоб допомогти вам писати JavaScript, який послідовно поводиться в клієнтському, серверному та рідному середовищах і простий у тестуванні.

Хоча він в основному використовується як інструмент управління станом разом з React, ви можете використовувати його з будь-якою іншою структурою або бібліотекою JavaScript. Він невеликий - 2 КБ (включаючи залежності), тому вам не доведеться турбуватися про те, що розмір активів вашої програми буде більшим.

За допомогою Redux статус вашої програми зберігається в магазині, і кожен компонент може отримати доступ до будь-якого статусу, який йому потрібен із цього магазину.

Нещодавно одна з найбільших дискусій у зовнішньому світі стосується Redux. Незабаром після виходу Redux став однією з найгарячіших тем обговорення.

Багато хто висловився "за", а інші вказували на проблеми.

Redux дозволяє вам керувати статусом вашої надбудови в одному місці та зберігати зміни до вмісту надбудови, більш передбачувані та відстежувані. Це полегшує думати про зміни, які з'являються у вашому додатку. Але всі ці переваги мають компроміси та обмеження. Хто може відчути, що він складає код шаблону, роблячи прості речі трохи приголомшливими; але це залежить від архітектурних рішень.

Одна проста відповідь на це запитання - ви знаєте, скільки вам потрібно Redux.

Якщо у вас є все, що вам потрібно, вам потрібно це вам не потрібно. Зазвичай це трапляється, коли ваш додаток розростається до такого масштабу, що управління умовами програми стає клопотом; і ви починаєте дивитися, щоб зробити це легко і просто.

State managment - це, по суті, спосіб полегшити спілкування та обмін даними між компонентами. Він створює відчутну структуру даних, яка представляє стан вашої програми, з якої ви можете читати та писати. Таким чином, ви можете побачити невидимі стани, працюючи з ними.

Більшість бібліотек, таких як React, Angular тощо, побудовані таким чином, що компоненти можуть управляти своїм станом внутрішньо, не потребуючи зовнішньої бібліотеки чи інструменту. Це добре для додатків з невеликою кількістю компонентів, але в міру того як додаток стає більшим, управління загальними між компонентами станами стає звичним явищем.

У програмі, де дані обмінюються між компонентами, може бути заплутаним насправді знати, де має проживати state. В ідеалі, дані в компоненті повинні містити лише один компонент, тому обмін даними між братами-компонентами стає важким.

Наприклад, у React, для обміну даними між братами та сестрами, state повиннен жити в батьківській складовій. Метод оновлення цього стану забезпечується батьківським компонентом і передається як пропс цим компонентам.

Ось простий приклад компонента входу React. Вхідні дані компонента входу впливають на те, що відображається його компонентом-братом, компонентом статусу:

```
class App extends React.Component{
  constructor(props) {
    super(props);
    // First the Parent creates a state for what will be passed
    this.state = { userStatus: "NOT LOGGED IN"}
    this.setStatus = this.setStatus.bind(this);
  }
```

А тепер уявіть, що відбувається, коли стан є загальним для компонентів, які знаходяться далеко від деревних компонентів.

Стейт повинен передаватися від одного компонента до іншого, якщо воно не потрапляє туди, куди потрібно.

По суті, стан веде до підключення до найближчого батьківського компонента та до наступного, поки не буде здійснено пошук пропозиції, спільної для обох компонентів, які потребують цього стану, а потім передано. Якщо ми використовуємо state, це менше очікується. Це також означає передачу непотрібних їм компонентів даних.

Зрозуміло, що управління стає брудним, оскільки додаток ускладнюється. Отже, вам потрібен стейт інструмент, такий як Redux, який полегшує підтримку цих штатів.

3.4 Redux у нашому веб-интерфейсу

Логіка стейт менеджера описана нижче, так у нас працює інтерфейс і зберігає дані які приходять з API (рис. 3.15).

3 lines (12 sloc) | 523 Bytes

```
import {applyMiddleware, combineReducers, compose, createStore} from "redux";
import thunkMiddleware from "redux-thunk";
import itemsReducer from "./itemsReducer";
import pendingsReducer from "./pendingsReducer";

let reducers = combineReducers({
    items:itemsReducer,
    pendings: pendingsReducer
});
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(reducers,  composeEnhancers(applyMiddleware(thunkMiddleware)));
window.__store__ = store;
export default store;
```

Рисунок 3.15 Стейт менеджер

Нижче ви можете побачити як ми обробляємо помилки в клієнт-серверній архітектурі за допомогою Redux (рис 3.16).

109 lines (103 sloc) | 3.2 KB

```
import {itemsAPI} from "../api/api";

const SET_ITEMS = 'SET_ITEMS'
const SET_STATUS = 'SET_STATUS'
const OK = 'OK';
const UNAUTHORIZED_PRIVATE_KEY = 'UNAUTHORIZED_PRIVATE_KEY'
const EMPTY_CHAIN = 'EMPTY_CHAIN'
const WRONG_CHAIN_HEIGHT_NUMBER = 'WRONG_CHAIN_HEIGHT_NUMBER'
const INVALID_ARGUMENT = 'INVALID_ARGUMENT'
const INVALID_TRANSACTION = 'INVALID_TRANSACTION'
const INVALID_SIGNATURE = 'INVALID_SIGNATURE'
const CARGO_ID_NOT_FOUND = 'Your cargo id not found'
const BLOCK_ALREADY_ADDED = 8
const WRONG_PREVIOUS_BLOCK_HASH = 9
const INVALID_BLOCK_TRANSACTIONS = 10
const INVALID_BLOCK = 11
const INVALID_BLOCK_HASH = 12
const INVALID_MERKLE_ROOT = 13
const CARGO_INFORMATION_ALREADY_EXISTS = "First information already exists, leave information field empty"
let initialState = {
    items: [],
    status: undefined
}

const itemsReducer = (state = initialState, action) => {
```

Рисунок 3.16 Код для обробки ошибки

Тут за допомогою цього шматка коду ми можемо отримувати всі дані з бекенд сервісу, в подальшому ми розпаковуємо ці дані і показуємо в Pendings сторінці.

Проміжне програмне забезпечення Redux Thunk дозволяє писати творців дій, які повертають функцію замість дії. Thunk може бути використаний для затримки відправлення дії або для відправки лише за умови дотримання певної умови (рис 3.17).

```
91  }
92  export const requestItems = (cargoId) => async (dispatch) => {
93      let response = await itemsAPI.getItems(cargoId)
94      errorHandler(response.data.result_code, dispatch);
95      dispatch(setItems(response.data.data))
96      console.log("good")
97  }
98
99
100 export const postItems = (dictionary) => async (dispatch) => {
101     let response = await itemsAPI.postItems(dictionary);
102     errorHandler(response.data.result_code, dispatch)
103     if (response.data.result_code === 0) {
104         dispatch(setStatus("Transaction sent"))
105     }
106     console.log(response)
107 }
108
109 export default itemsReducer;
```

Рисунок 3.17 Робота Redux-Thunk

По-перше, редуктор повинен прослухати надіслані дії та оновити статус вашої програми в магазині.

Тоді ваші компоненти (або подані матеріали) повинні передплатити магазин і оновити свій локальний статус після оновлення магазину.

3.5 Процес майнингу у системі

Як і в будь-якому блокчейне потрібна потужність яка буде створювати транзакції, ми зробили логіку майнінг, однак це не схожий на майнінг біткойнов, тут йде мінімальні зусилля щоб прокласти нову транзакцію (рис. 3.18).

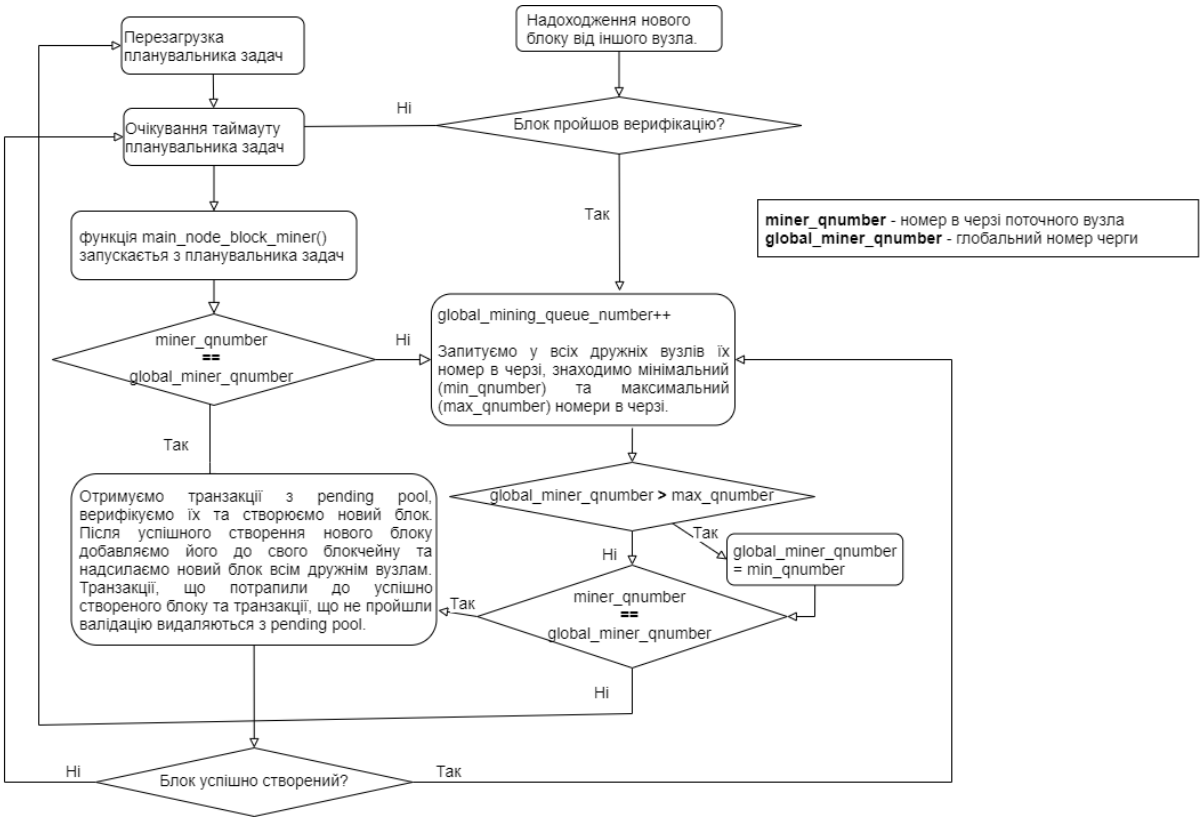


Рисунок 3.18 Схема реалізації майнінга за чергою.

Алгоритм розроблений в рамках даної дипломної роботи.

У мережі біткойнів середній час розв’язування головоломки майнерами становить 10 хвилин (Litecoin ~ 2,5 хвилини, Ethereum ~ 15 секунд). Імовірність того, що два або більше майнерів успішно розгадають головоломку приблизно за один і той же час і передають новий блок в мережу, пропорційна частоті створення нових блоків). Це суперечить стабільності мережі, де ми хочемо мінімізувати мережеві пробки.

Чим довший середній час створення нового блоку, тим більша ймовірність транзакцій охопить усіх користувачів. Це сприяє підвищенню стабільності мережі, запобігаючи появі форків.

Безпека та безпека

Другий аспект майнінгу - збільшення незворотності транзакції. Вилучення нового блоку вимагає енергії для вирішення головоломки. Транзакції всіх внутрішніх блоків захищають всю енергію, витрачену на вилучення до них блоків. Чим старша одиниця, тим менше шансів, що вона осиротіла. Чим старший пристрій, тим більше енергії буде потрібно для модифікації приладу. ***)

Чесність

Винагороди, отримані від транзакцій на монетній базі, стимулюють учасників мережі (наприклад, нові біткойни) і тому, що вони стягуються за транзакції, включені до відповідного блоку. Гонка за блок приносить справедливість розподілу стимулів між майнерами пропорційно потужності, яку вони витрачають на захист блокчейну. Чесність також є ще однією перевагою з точки зору децентралізації та безпеки, але також не може бути розвинена для ясності зараз.

Доказ роботи забезпечує високий рівень безпеки, безпеки та незворотності, необхідний в анонімному громадському середовищі. Ціна цього - високі енергетичні витрати та обмежена пропускна здатність транзакцій.

Давайте оцінимо вищі властивості у приватному та обмеженому блокчейні.

Кількість відвідувачів ділової мережі, як правило, на порядок нижче, і якщо існує пряма видимість мережі. Тому вам не потрібні великі часові вікна, щоб забезпечити розповсюдження транзакцій по всіх мережах. Повідомлення про створення нових блоків за допомогою PoW не потрібно або не досягати цього рівня для досягнення стабільності.

У дозволених та приватних блокчейнах відомі особисті учасники. Це обмежено для тих, кому дозволено брати участь у мережі, дотримуватися консенсусного протоколу та вести спільні книги. Зазвичай не існує власницького токена або стимулу для мотивації членів приєднатися та виконувати майнінг. Для розподілу стимулів взагалі не потрібна справедливість.

Оскільки учасники не є анонімними, мережа, як правило, не підходить для ворожих громадських середовищ Інтернету, а вартість незмінності слабка. Блокчейн не повинен бути захищений величезними енергетичними витратами. Для незмінності, як правило, достатньо цілісності через хеш-ланцюжок та копії, що належать різним сторонам.

Висновки за розділом 3

Отже, проаналізувавши програму розроблену в даній дипломній роботі, що побудоване на базі блокчейну, RFID міток та веб-додатку, можна побачити, що структура додатку надзвичайно проста, та легко розширювана, що дозволяє продовжувати розробку та вдосконалювати алгоритми, які він використовує.

Вся ця структура программи була розроблена для захисту користувачів від коррупцій. А також орієнтована на те, щоб подальша розробка була легкою та ефективною.

Базуючись на своїй незалежній оцінці, розроблена програма достовірно визначає потенційно небезпечні проблеми логістики, та повідомляє користувача о випадках при перевозках, що може вчасно отреагувати запобігти витоку товару чі коррупцій.

ВИСНОВКИ

В процесі виконання дипломної роботи було розглянуто: теоретичні основи побудови приватного блокчейну, його API та користувацького інтерфейсу, види радіо міток та особливості реалізації їх зв’язку з блокчейном.

В результаті виконання дипломної роботи була побудована програмна система, за допомогою якої можна досліджувати переміщення будь-якого вантажу по логістичному ланцюгу будь-якого підприємства або організації.

Розроблено приватний блокчейн, його API та користувацький інтерфейс у вигляді SPA, програму для зчитування даних з NFC-мітки та реалізацію зв’язку цих даних блокчейном.

За останні роки нові технології стрімко набирають оберти.

Логістична сфера, як одна з найважливіших галузей економіки, потребує свіжого повітря у вигляді впровадження найновіших технологічних рішень. Тому є досить важливим проводити подібні дослідження та вносити свій вклад в розвиток світової економіки.

Всі ми знаємо що зараз в світі дуже багато корупції і проблем з нелегальними оборотами.

В ході дипломної роботи я зробив програму яка зможе вирішити цю проблему раз і назавжди.

В першому розділі роботи було проаналізовано теоретичні питання блокчейну, RFID міток , що є складною системою, яка стала все популярною і продовжує надалі розширювати свою популярність. Вся відповідальність за безпечну передачу інформації лежить на користувача, не на систему.

У другому розділі даної дипломної роботи був розглянутий технологічний стек, структура та архитектура програми, виведена формула розрахунку, та розроблена блокчейн программа для доеднаня до интерфейсу веб користувача. Були використані відомі технології Python, Django, SPI, HTML, CSS, JavaScript, а також сучасні React, Npm (Node Package Manager), Webpack, Redux, Axios, Redux-Thunk.

В третьому розділі дипломної роботи було показана робота интерфейсу. З цього було визначено, що програма надзвичайно складна, але може бути легко розширювана, що дозволяє доволі просто продовжувати розробку та вдосконалювати алгоритми, які використовує программа цієї роботи. Розширення легко інтегрується з системою контролю версій git, що значно спрощує розробку і є необхідною складовою у сучасному світі програмного забезпечення.

Головна мета була реалізована - це створення складної, але надійної системи для захисту людей від подалі злочинності в світі. Ця програма дуже ефективна бо її майже що неможливо зламати. У неї було витрачено купа годин розробки та старань. Звичайно ідеальної системи безпеки не буває, проте ми повинні завжди прагнути до цього

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Приклади технології блокчейн в логістиці. [Електронний ресурс]– Режим доступу: https://merehead.com/ru/blog/top-benefits-blockchain-logistics-use-cases/

2. Технологія RFID, мітки, зчитувачі, їх застосування. [Електронний ресурс] –Режим доступу: https://realtrac.com/ru/company/blog/princip_raboty_tehnologii_rfid_i_ee_primenenie/

3. Дерево Меркла [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%9C%D0%B5%D1%80%D0%BA%D0%BB%D0%B0

4. EllipticCurveDigitalSignatureAlgorithm [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

5. Серіалізація [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F

6. RFID-модуль RC522[Електронний ресурс] – Режим доступу: https://3d-diy.ru/wiki/arduino-moduli/rfid-modul-rc522/

7. RaspberryPi [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Raspberry_Pi

8. Шнайер Брюс. «Прикладная криптография. Протоколы, алгор.на С». : Диалектика, 2017. 1040 с.

9. Павел Кравченко, Богдан Скрябин, Оксана Дубинина «Блокчейн и децентрализованные системы. Часть 1»: DistributedLab, 2018. 470 с.

10. React [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/React

11. Технологія Axios [Електронний ресурс] – Режим доступу: https://github.com/axios/axios

12. Axios: promise based HTTP client for the browser and node.js [Електронний ресурс] – Режим доступу: https://www.npmjs.com/package/axios

13. Where does Chrome store its cookie file? [Електронний ресурс] – Режим доступу: https://superuser.com/questions/459426/where-does-chrome-store-its-cookie-file

14. An Introduction to Content Security Policy – Mike West [Електронний ресурс] – Режим доступу: https://www.html5rocks.com/en/tutorials/security/content-security-policy/

15. API method to retrieve all cookies from a single cookie store - Google [Електронний ресурс] – Режим доступу: https://developer.chrome.com/extensions/cookies - method-getAll

16. APACHE LICENSE, VERSION 2.0 [Електронний ресурс] – Режим доступу: http://www.apache.org/licenses/LICENSE-2.0

17. Технологія JavaScript [Електронний ресурс] – Режим доступу: https://javascript.info/intro

18. Технологія Babel [Електронний ресурс] – Режим доступу: https://babeljs.io/

19. Технологія npm [Електронний ресурс] – Режим доступу: https://www.npmjs.com/

20. Технологія Webpack [Електронний ресурс] – Режим доступу: https://webpack.js.org/

21. Модель OSI і TCP/IP: Офіційний сайт компанії Cisco [Електронний ресурс] – Режим доступу: Cisco - Global Home Page

22. HTTP Protocol [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/http/http_methods.htm

23. HTTPS Protocol [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/security_testing/https_protocol_basics.htm

24. History of Web Browsers [Електронний ресурс] – Режим доступу: https://midlothianweb.com/history-of-web-browsers/

25. History of Web Applications: Офіційний сайт компанії Coursera [Електронний ресурс] – Режим доступу: https://www.coursera.org/lecture/web-app/video-3-evolution-of-web-apps-yghKM

26. Git is a free and open source distributed version control system [Електронний ресурс] – Режим доступу: https://git-scm.com/

27. Redux A Predictable State Container for JS Apps [Електронний ресурс] – Режим доступу: https://redux.js.org/

28. Redux Thunk thunk middleware for Redux [Електронний ресурс] – Режим доступу: https://github.com/reduxjs/redux-thunk

29. Material-UI: A popular React UI framework [Електронний ресурс] – Режим доступу: https://material-ui.com/

30. Sass (Syntactically Awesome Stylesheets) [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Sass

ДОДАТОК А.

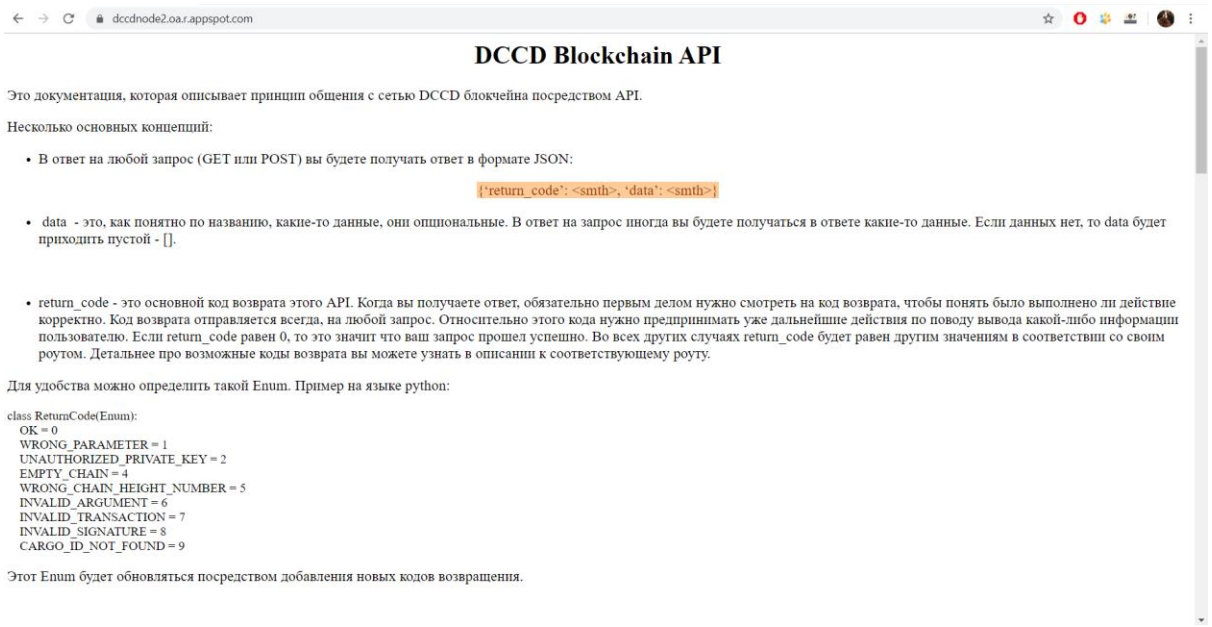
Реалізація генерації приватного ключа.

```
class KeyGenerator:
    def __init__(self):
        self.POOL_SIZE = 256
        self.KEY_BYTES  = 32
        self.CURVE_ORDER = int('FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE
6AF48A03BBFD25E8CD0364141',  16)
        self.pool = [0] * self.POOL_SIZE
        self.pool_pointer = 0
        self.prng_state = None
        self.__init_pool()

    def seed_input(self, str_input):
        time_int  = int(time.time())
        self.__seed_int(time_int)
        for char in str_input:
            char_code = ord(char)
            self.__seed_byte(char_code)

    def generate_key(self):
        big_int  = self.__generate_big_int()
        big_int  = big_int  % (self.CURVE_ORDER - 1)
        big_int  = big_int  + 1
        key = hex(big_int)[2:]
        key = key.zfill(self.KEY_BYTES  * 2)
        return key
```

```
def __init_pool(self):
    for i in range(self.POOL_SIZE):
        random_byte = secrets.randbits(8)
        self.__seed_byte(random_byte)
    time_int  = int(time.time())
    self.__seed_int(time_int)

def __seed_int(self, n):
    self.__seed_byte(n)
    self.__seed_byte(n >> 8)
    self.__seed_byte(n >> 16)
    self.__seed_byte(n >> 24)

def __seed_byte(self, n):
    self.pool[self.pool_pointer] ^= n & 255
    self.pool_pointer += 1
    if self.pool_pointer >= self.POOL_SIZE:
        self.pool_pointer = 0

def __generate_big_int(self):
    if self.prng_state is None:
        seed = int.from_bytes(self.pool, byteorder='big', signed=False)
        random.seed(seed)
        self.prng_state = random.getstate()
    random.setstate(self.prng_state)
    big_int  = random.getrandbits(self.KEY_BYTES  * 8)
    self.prng_state = random.getstate()
    return big_int
```

ДОДАТОК Б

Опис API розробленого блокчейну, що можна побачити у формі HTML сторінки.



dccdnode2.oa.r.appspot.com

DCCD Blockchain API

Это документация, которая описывает принцип общения с сетью DCCD блокчейна посредством API.

Несколько основных концепций:

- В ответ на любой запрос (GET или POST) вы будете получать ответ в формате JSON:

{‘return_code’: <smth>, ‘data’: <smth>}

- data - это, как понятно по названию, какие-то данные, они опциональные. В ответ на запрос иногда вы будете получаться в ответе какие-то данные. Если данных нет, то data будет приходить пустой - [].

- return_code - это основной код возврата этого API. Когда вы получаете ответ, обязательно первым делом нужно смотреть на код возврата, чтобы понять было выполнено ли действие корректно. Код возврата отправляется всегда, на любой запрос. Относительно этого кода нужно предпринимать уже дальнейшие действия по поводу вывода какой-либо информации пользователю. Если return_code равен 0, то это значит что ваш запрос прошел успешно. Во всех других случаях return_code будет равен другим значениям в соответствии со своим роутом. Детальнее про возможные коды возврата вы можете узнать в описании к соответствующему роуту.

Для удобства можно определить такой Enum. Пример на языке python:

```
class ReturnCode(Enum):
    OK = 0
    WRONG_PARAMETER = 1
    UNAUTHORIZED_PRIVATE_KEY = 2
    EMPTY_CHAIN = 4
    WRONG_CHAIN_HEIGHT_NUMBER = 5
    INVALID_ARGUMENT = 6
    INVALID_TRANSACTION = 7
    INVALID_SIGNATURE = 8
    CARGO_ID_NOT_FOUND = 9
```

Этот Enum будет обновляться посредством добавления новых кодов возвращения.

DCCD Blockchain API

Це документація, яка описує принцип спілкування з мережею DCCD блокчейна за допомогою API.

Кілька основних концепцій:

У відповідь на будь-який запит (GET або POST) ви будете отримувати відповідь в форматі JSON:

{‘return_code’: <smth>, ‘data’: <smth>}

data - це, як відомо за назвою, якісь дані, вони опціональні. У відповідь на запит іноді ви будете виходити у відповіді якісь дані. Якщо даних немає, то data буде приходити порожньою- [].

return_code - це основний код повернення цього API. Коли ви отримуєте відповідь, обов'язково насамперед потрібно дивитися на код повернення, щоб зрозуміти було виконано дія коректно. Код повернення відправляється завжди, на будь-який запит.

Щодо цього коду потрібно робити вже подальші дії з приводу виведення будь-якої інформації користувачеві. Якщо return_code дорівнює 0, то це означає, що ваш запит пройшов успішно. У всіх інших випадках return code буде дорівнює іншим значенням відповідно до свого раутом. Детальніше про можливі коди повернення ви можете дізнатися в описі до відповідного Рауса.Для удобства можно определить такой Enum. Пример на языке python:

```
class ReturnCode(Enum):
    OK = 0
    WRONG_PARAMETER = 1
    UNAUTHORIZED_PRIVATE_KEY  = 2
    EMPTY_CHAIN = 4
    WRONG_CHAIN_HEIGHT_NUMBER = 5
    INVALID_ARGUMENT  = 6
    INVALID_TRANSACTION = 7
    INVALID_SIGNATURE  = 8
    CARGO_ID_NOT_FOUND = 9
```

Цей Enum буде оновлюватися за допомогою додавання нових кодів повернення.

ROUTES

/miner/queue/number [GET] – дозволяє дізнатися який номер в черзі Майнінг знаходиться цей вузол. Цей функціонал служить для коректного визначення черги Майнінг між усіма вузлами в мережі.

Return: {‘return_code’: 0, ‘data’: <miningnumber>}

/newblock [POST] - дозволяє запропонувати вузловий блок. Це використовує вузол після того як створить новий блок і починає розповідати про це всім своїм дружнім нодам. Вузол, який отримав такий запит повинен так само розповісти всім своїм сусідам, в разі якщо у нього ще не було такого блоку.

Return: {‘return_code’: 0, ‘data’: []}

/addnode [POST] - дозволяє додати новий дружній вузол поточного вузла. Під час додавання йде перевірка на "сумлінність". Перевіряється ланцюжок цього вузла на відповідність протоколу та нашої ланцюжку. Якщо вузол виявляється некоректним, то додавання не буде виконано. // в процесі

Return: {‘return_code’: 0, ‘data’: []}

/nodes [GET] - дозволяє отримати список дружніх вузлів поточного вузла.

Return: {‘return_code’: 0, ‘data’: <nodes_list>}

/transactions/pendings [GET] - повертає список транзакцій з меморіпула. Це ті транзакції, які були створені користувачем, пройшли перевірку і чекають поки їх заберуть в блок.

Return: {‘return_code’: 0, ‘data’: []}

/chain?height=<height> [GET] - дозволяє отримати ланцюжок певної висоти, починаючи з кінця ланцюга (з останнього змайненного блоку), поточного вузла.

Можливі відповіді в return_code:

EMPTY_CHAIN = 4 - в цьому вузлі немає жодного блоку в ланцюзі.

WRONG_CHAIN_HEIGHT_NUMBER = 5 – занадто велике число ланцюга (довжина ланцюга менше зазначеної висоти)

INVALID_ARGUMENT = 6 – введений не дійсний аргумент висоти, або взагалі не введений.

Return: {‘return_code’: <code>, ‘data’: []} Якщо ж ваш запит коректний, ви отримаєте у відповідь ви запросили кількість блоків:

Return: {‘return_code’: 0, ‘data’: <block_list>}

/chain/length [GET] – повертає довжину ланцюга поточного вузла.

Return: {‘return_code’: 0, ‘data’: <len>}

/find?<arg_name>=<arg_data>[GET] – повертає пошукові дані, в залежності від аргументу.

Можливі аргументи:

cargo_id = повертає список транзакцій із зазначеним cargoid в форматі JSON в поле data.

block_by_height = повертає блок по його номеру в форматі JSON в поле data.

Можливі відповіді в return_code:

EMPTY_CHAIN = 4 - в цього вузла немає жодного блоку в ланцюзі.

WRONG_CHAIN_HEIGHT_NUMBER = 5 – занадто великий номер блоку (довжина ланцюга менше зазначеної висоти)

INVALID_ARGUMENT = 6 – введений не дійсний аргумент, або взагалі не введений.

CARGO_ID_NOT_FOUND = 9 – введений cargo_id не найден.

Return: {‘return_code’:<code>, ‘data’:[]}

Якщо ж ваш запит коректний, ви отримаєте у відповідь блок запитаної висоти:

Return: {‘return_code’:0, ‘data’:<block>}

/transaction/new?<arg_name>=<arg_data> [POST] – пропонує вузлу нову підписану транзакцію.

Підпис відбувається на стороні клієнта, а на роут відправляється вже підписана транзакція, або серіалізовані або в форматі словника.

Відповідно транзакцію можна відправити в таких аргументах:

serialized - серіалізовані транзакція

dictionary - транзакція в форматі словника

Можливі відповіді в <return_code>:

INVALID_ARGUMENT = **6** – введений не дійсний аргумент, або взагалі не введений.

INVALID_TRANSACTION = **7** – відсутня транзакція в аргументі

UNAUTHORIZED_PRIVATE_KEY = **2** - приватний ключ, яким підписана транзакція не включений в список авторизованих приватних ключів для підпису транзакцій.

INVALID_SIGNATURE = **8** – цифровий підпис некоректний. Можливо, ваша транзакція була змінена після підпису.

Якщо ж ваш запит коректний, транзакція попадет в мемори пул поточної Ноди і буде чекати поки її заберуть в блок. Ви відповідно отримаєте таку відповідь:

Return: {‘return_code’: 0, ‘data’: []}

ДОДАТОК В

Код Веб-інтерфейсу.

App.js:

```
import React from 'react';
import './App.css';
import {Route, Switch} from "react-router-dom";
import {Main} from "./Main";
import {Find} from "./components/FindCargo/Find/Find";
import {NewTransaction} from
"./components/NewTransaction/NewTransaction/NewTransaction";
import {Pendings} from './components/Pendings/Pendings/Pendings'
const App = (props) => {
    return (
        <div className='app-wrapper'>
            <Switch>
                <Route exact={true} path='/' render={() => <Main/>}/>
                <Route exact={true} path='/find' render={() => <Find/>}/>
                <Route exact={true} path='/new' render={() =>
<NewTransaction/>}/>
                <Route exact={true} path='/pendings' render={() =>
<Pendings/>}/>
            </Switch>
        </div>
    )
}

export default App;
```

Find.js

```
import React from "react";
```

```
import style from "./Find.module.css"
import {Header} from "../../Header/Header";
import {Footer} from "../../Main/Footer/Footer";
import TextField from '@material-ui/core/TextField';
import Button from "@material-ui/core/Button";
import {useFindForm} from "./useFindHook";

const FindResult = (props) => {
    let a = new Date(props.timestamp * 1000);
    let months =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'];
    let year = a.getFullYear();
    let month = months[a.getMonth()];
    let date = a.getDate();
    let hour = "0" + a.getHours();
    let min = "0" + a.getMinutes();
    let sec = "0" + a.getSeconds();
    let time = date + ' ' + month + ' ' + year + ' ' + hour.substr(-2) + ':' +
min.substr(-2) + ':' + sec.substr(-2) ;
    return (
        <div className={style.mainResult}>
            <h3>Your CargoId: <p>{props.cargo_id}</p></h3>
            <h3>Information: <p>{props.information}</p></h3>
            <h3>Temperature: <p>{props.temperature}</p></h3>
            <h3>Timestamp: <p>{time}</p></h3>
            <h4>Public Key: <p>{props.public_key}</p></h4>
            <h4>Signed Hash: <p>{props.signed_hash}</p> </h4>
        </div>
    )
}
```

```
export const Find = (props) => {
    const {inputs, handleInputChange, handleSubmit, items, status} =
useFindForm({cargoId: ''});
    let itemsElements = items.map(items => <FindResult
cargo_id={items.cargo_id}
                                        information={items.information}
                                        public_key={items.public_key}
                                        signed_hash={items.signed_hash}
                                        temperature={items.temperature}
                                        timestamp={items.timestamp}/>
    )
    return (
        <div>
            <Header/>
            <div className={style.find}>
                <form onSubmit={handleSubmit}>
                    <h1>Find your <span>Cargo Id</span></h1><br/>
                    <TextField
                        variant="outlined"
                        margin="normal"
                        required
                        className={style.field}
                        value={inputs.cargoId}
                        onChange={handleInputChange}
                        id="cargoId"
                        label="CargoId"
                        name="cargoId"
                        autoFocus
                    /><br/>
                    <Button
```

```
                    type="submit"
                    variant="contained"
                    color="primary"
                  >
                    Find
                  </Button>
                </form>
                <div className={style.status}>
                  <h2>{status}</h2>
                </div>
                {itemsElements}
              </div>
              <Footer/>
            </div>
        )
    }
```

Transaction.js

```
    import React from "react";
    import style from "./NewTransaction.module.css"
    import {Header} from "../../Header/Header";
    import {Footer} from "../../Main/Footer/Footer";
    import TextField from '@material-ui/core/TextField';
    import Button from "@material-ui/core/Button";
    import {useTransactionForm} from "./useNewTransactionHook";
    import TextareaAutosize from "@material-ui/core/TextareaAutosize";

    export const NewTransaction = (props) => {
        const {inputs, handleInputChange, handleSubmit, status, error} =
    useTransactionForm({
```

```
    cargoId: '',
    privateKey: '',
    information: ''
  });
  return (
    <div>
      <Header/>
      <div className={style.find}>
        <form onSubmit={handleSubmit}>
          <h1>Make your <span>New
Transaction</span></h1><br/>
          <div className={style.field}>
            <TextField
              variant="outlined"
              margin="normal"
              required
              className={style.field}
              value={inputs.cargoId}
              onChange={handleInputChange}
              id="cargoId"
              label="CargoId"
              name="cargoId"
              autoFocus
            /><br/>
            <TextareaAutosize onChange={handleInputChange}
value={inputs.information}
                              className={style.textArea} rowsMin={16}
id="information"
                              label="Information"
                              name="information"/>
```

```
                    <TextField
                        variant="outlined"
                        margin="normal"
                        required
                        className={style.field}
                        onChange={handleInputChange}
                        value={inputs.privateKey}
                        id="privateKey"
                        label="WIF-format private key"
                        name="privateKey"
                        autoFocus
                    /><br/>
                </div>
                <Button
                    type="submit"
                    variant="contained"
                    color="primary"
                >
                    SEND
                </Button>
            </form>
            <div className={style.status}>
                <h2>{status === "Transaction sent" ? <div
className={style.green}>{status}</div> : status }</h2>
                <h2>{error}</h2>
            </div>
        </div>
        <Footer/>
    </div>
  )
```

```
}
```

Pending.js

```
import React from "react";
import style from "./Pendings.module.css"
import {usePendingsHook} from "./usePendingsHook";
import {Header} from "../../Header/Header";
import Button from "@material-ui/core/Button";

export const PendingsResult = (props) => {
    let a = new Date(props.timestamp * 1000);
    let months =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'];
    let year = a.getFullYear();
    let month = months[a.getMonth()];
    let date = a.getDate();
    let hour = "0" + a.getHours();
    let min = "0" + a.getMinutes();
    let sec = "0" + a.getSeconds();
    let time = date + ' ' + month + ' ' + year + ' ' + hour.substr(-2) + ':' +
min.substr(-2) + ':' + sec.substr(-2) ;
    return (
        <div className={style.mainResult}>
            <h3>Your CargoId: <p>{props.cargo_id}</p></h3>
            <h3>Information: <p>{props.information}</p></h3>
            <h3>Temperature: <p>{props.temperature}</p></h3>
            <h3>Timestamp: <p>{time}</p></h3>
            <h4>Public Key: <p>{props.public_key}</p></h4>
            <h4>Signed Hash: <p>{props.signed_hash}</p></h4>
        </div>
```

```
    )
}


export const Pendings = (props) => {
    const {handleSubmit, pendings} = usePendingsHook();
    let pendingsElements = pendings.map(pendings => <PendingsResult
cargo_id={pendings.cargo_id}

information={pendings.information}

public_key={pendings.public_key}

signed_hash={pendings.signed_hash}

temperature={pendings.temperature}

timestamp={pendings.timestamp}
    />)
    return (
        <div>
            <Header/>
            <div className={style.find}>
                <Button
                    type="submit"
                    variant="contained"
                    color="primary"
                    onClick={handleSubmit}
                >
                    Refresh
```

```
            </Button>
            {pendingsElements}
          </div>
        </div>
    )
}
```