Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ТИПИ КОНСЕНСУСУ В БЛОКЧЕЙНІ

Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки" 6.050122

Керівник курсової роботи
к.т.н., доц. _____
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2020 р.

Виконав студент _____
_____
(прізвище та ініціали)
"____" _____ 2020 р.

Київ 2020

Тема : Типи консенсусу в блокчейні

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	04.11.2019	
2.	Огляд технічної літератури за темою роботи.	18.11.2019	
3.	Ознайомлення з літературою.	25.11.2019	
3.	Вивчення принципів роботи мережі блокчейн.	30.12.2019	
4.	Проведення аналізу використання типів консенсусу.	07.01.2020	
5.	Детальний розгляд принципів функціонування методів консенсусу.	15.01.2020	
6.	Початок проектування основної структури практичної частини.	11.02.2020	
7.	Програмування реалізації практичної частини.	20.03.2020	
8.	Створення слайдів для доповіді та написання доповіді.	22.04.2020	
8.	Аналіз отриманих результатів з керівником, закінчення роботи над практичною частиною, закінчення роботи над доповіддю.	25.04.2020	
10.	Корегування практичної роботи з керівником.	1.05.2020	
11.	Остаточне оформлення доповіді та слайдів.	8.05.2020	
12.	Захист курсової роботи.	18.05.2020	

АНОТАЦІЯ

У даній роботі розглядається новітня технологія «блокчейн» та її основні застосування, включаючи найбільш відоме та широко використовуване – криптовалюти. Проводиться детальний огляд основних алгоритмів, що забезпечують консенсус під час обміну криптовалютами, та надаються деталі завдання, що призначене для полегшення практичного використання криптовалют.

Зміст

ВСТУП

Постійний прогрес людства у будь-якій сфері зазвичай супроводжується спробами використання нових технологій для полегшення повсякденних справ, і сфера комп'ютерних технологій не є виключенням. Однією з таких нових технологій є блокчейн, технологія відкритого пов'язаного розподіленого списку, процес функціоналу якої полягає у розв'язанні проблеми «візантійської стійкості» через отримання криптографічного хешу учасниками системи у з'єднанні за допомогою peer-to-peer протоколу.

Хоч прямий опис блокчейну і є досить складним, ця технологія знайшла місце у багатьох сферах та, зокрема, у сфері віртуальних грошей – криптовалют. Природно, що в процесі адаптації цієї технології були створені нові алгоритми, які у перспективі можуть зробити блокчейн ще більш корисним або навіть дати початок новим інноваційним ідеям, що будуть фундаментом технологій майбутнього. Розглянути їх усі практично неможливо, тому ця робота зупиниться лише на загальному описі основи функціонування блокчейну – алгоритмах консенсусу.

Тема цієї роботи – детальне дослідження, опис та порівняння різних типів алгоритмів консенсусу, а також короткий опис їх застосувань на момент її написання. Кожен з типів буде розглянутий з точки зору його порівняної швидкості, пропускної здатності, масштабованості, використання системних ресурсів, безпеки, анонімності та історії використання.

Ця робота ставить на меті дослідження вертикального прориву під назвою блокчейн – встановлення причин його популярності, потенційних та існуючих галузей застосування, в особливості галузі криптовалют як основної та найбільш розвиненої, методів його використання на прикладі типів консенсусу та наведення прикладів його інтеграції.

РОЗДІЛ 1: Основи технології блокчейн

1.1 Поняття блокчейну

Блокчейн (від англ. block chain – ланцюг блоків) – це зв'язаний список певних записів, або блоків, що функціонує з допомогою технології криптографічного хешування. Кожен з блоків у блокчейні містить у собі певну інформацію у залежності від використання цієї технології, а також захешоване посилання на попередній блок та позначення часу транзакції [1].

Така структура дозволяє зробити інформацію у блокчейні досить складною для модифікації навіть у звичайній системі, оскільки для заміни певного блоку необхідно замінити усі блоки, що посилаються на нього. Проте, як варто зауважити, система у якій працює блокчейн не є звичайною, оскільки ця технологія функціонує за принципом «open distributed ledger», або «відкритої розподіленої книги». Це означає що усі записи зберігаються на цілій мережі пов'язаних між собою комп'ютерів, що надалі унеможливлює зміну даних – для цього необхідний консенсус більшості учасників.

Блокчейн працює за технологію P2P (peer-to-peer). Усі «піри» (окремі комп'ютери) мають певні права в системі, на них зберігається певна частина інформації. Серед них окремо визначаються «повні ноди», що зберігають усю інформацію. У залежності від конкретного використання блокчейну, можуть існувати додаткові категорії користувачів, у яких є право ухвалювати транзакції або приймати певні рішення про розвиток блокчейну.

Завдяки вищезазначеним факторам, блокчейн є розподіленою структурою, що забезпечує високу так звану «візантійську стійкість» – стійкість даних до зміни унаслідок «неправильного» поводження користувачів системи, наприклад, технічних помилок або шахраювання. Як наслідок, ця технологія є дуже корисною для зберігання та змін відкритої інформації без зменшення рівню її захисту.

Також однією з позитивних сторін цієї структури є її децентралізованість. Оскільки мережа здебільшого базується на рівності внеску усіх користувачів у збереження даних, в неї практично немає певного «центру» атаки, що значно ускладнює атаки на стабільність системи. Блокчейн був першим методом вирішення проблеми «double

spending» (букв. «подвійної оплати») що не потребує центрального серверу обчислень або сертифікатів завірення третьою особою.

Блокчейн був винайдений особою (або групою осіб), що використовують прізвисько Сатоші Накамото, в 2008 році, щоб служити головним журналом транзакцій криптовалюти Bitcoin. Особа Сатоші Накамото залишається невідомою і по сьогодні.

Існує три типи блокчейнів:

- Публічні блокчейни.
- Приватні блокчейни.
- Гібридні (консорціумні) блокчейни.

Публічні блокчейни не обмежують участь у них. Будь-хто, хто має технічне обладнання необхідне для виконання програми, що використовує блокчейн, може взаємодіяти з даним блокчейном та бути частиною алгоритму консенсусу. Авторизація чи ідентифікація користувачів відсутня, що дозволяє повну анонімність, та платформа є повністю децентралізованою. Криптовалюти переважно використовують цей тип блокчейну.

Приватні блокчейни максимально обмежують участь у них. Дозвіл центрального серверу необхідний як для читання, так і для запису блоків, та весь блокчейн повністю належить одній організації. Анонімності у блокчейнах цього типу немає та бути не може. Такі блокчейни корисні для збереження операцій всередині організацій.

Гібридні блокчейни є серединою між приватними та публічними блокчейнами. Вони можуть не мати систем ідентифікації (проте завжди мають авторизацію) користувачів, зазвичай належать групі осіб, і усі учасники що мають дозвіл на надання нових записів є відомими для інших. Такі блокчейни використовуються для обміну даними між кількома організаціями або просто певними категоріями людей.

1.2 Практичне застосування блокчейну

«Практичний наслідок [… це те…] що вперше з'явився спосіб одному користувачеві Інтернету передати унікальний фрагмент цифрової власності іншому користувачеві Інтернету, таким чином, щоб передача гарантовано була безпечною та надійною, щоб всім був відомо, що передача відбулася, і щоб ніхто не міг оскаржити законність передачі. Наслідки цього прориву важко переоцінити.» [2] – Марк Андрессен.

Завдяки вищеописаним характеристикам блокчейну – його відкритості, захищеності та децентралізації – він набув надзвичайної популярності у сучасному світі комп'ютерних технологій. Як приклад, деякі компанії отримали приріст у ціні акцій у розмірі до 42500% просто змінивши свою назву так, щоб вона включала у себе терміни «біткоїн» чи «блокчейн» [3].

Цей рівень популярності, звичайно ж, означає що було створена надзвичайна кількість нових ідей по використанню нової технології. Як це часто стається, деякі з них виявились провальними (або виявляться провальними у близькому майбутньому), але досить велика частина все ж продовжує існувати і розвиватись і зараз.

Наразі блокчейн використовується або планується використовуватись у таких галузях:

- Криптовалюти: переважна більшість криптовалют що є у циркуляції на час написання цієї роботи використовують блокчейн для збереження транзакцій та запобігання шахрайства.
- Фінансові сервіси: значна кількість банківських систем розпочинають перехід на розподілені банківські книги, що базуються на блокчейн-технологіях.
    - Bitwala, німецька блокчейн-банкінгова компанія, що була заснована у 2015 році, у грудні 2018 року запустила перше в Європі регульоване блокчейн-банківське рішення, яке дозволяє користувачам керувати як своїми депозитами в Bitcoin, так і євро в одному місці, забезпечуючи безпеку та зручність німецького банківського рахунку. [4]
- Відеоігри: існує ціла категорія «блокчейн відеоігр», що використовують блокчейн для збереження стану предметів, що належать користувачу.
- Медицина: за допомогою смарт-контрактів стає можливою швидка і надійна передача приватної інформації між лікарями та клієнтами. Також деякі медичні установи використовують блокчейн для збереження даних про пацієнтів для поширення знань про їх стан здоров'я та медичні потреби без необхідності використання медичних карт.
- Сільське господарство: блокчейн може використовуватись у поєднанні з IoT-пристроями для передачі та збереження даних про стан сільськогосподарських

угідь для фіксації і виправлення проблем на певній місцевості та розповсюдження досвіду для підвищення ефективності.

- Інтернет речей (IoT): існують сервіси що використовують блокчейн для підвищення стійкості IoT-девайсів до злому.
- Державні служби: планується використання блокчейн-технологій для забезпечення безпечних онлайн-голосувань та збереження приватних документів та даних громадян.
- Вантажні перевезення: технологія розподіленого журналу використовується багатьма компаніями (зокрема, DHL, Maersk та ShipChain) для відстеження стану та прогресу перевезень.
- Інтелектуальна власність: завдяки можливості швидко, безпечно та надійно зареєструвати інформацію про інтелектуальну власність та час її встановлення значно зменшується можливість потенціальних диспутів про права на неї. Також зменшується негативний внесок так званих «патентних тролів» – людей, що володіють патентами на певну інтелектуальну власність лише з метою виграшу грошей у суді коли ці патенти порушуються.
- Нерухомість: завдяки смарт-контрактам та відкритій базі даних про репутацію орендаторів та їх клієнтів можливо зробити процес оренди та купівлі нерухомості значно швидшим та зручнішим.
- Благодійність: за допомогою блокчейну стає можливим значне збільшення відкритості та надійності транзакцій, зменшення витрат на операцію благодійних організацій, та захист від шахрайства.
- Юридичний сектор: для юридичного сектору документація є критичною частиною будь-якого процесу, що лише збільшує витрати часу та грошей через її громіздкість та неефективність. Безпечне збереження критичних контрактів та документів може бути проблематичним, враховуючи помилки людських ресурсів. Смарт-контракти дозволяють позбавитись цих проблем, роблячи роботу юристів значно більш ефективною або й взагалі забираючи необхідність людської участі у деяких частинах юридичної роботи.

- Технічні засоби: зберігання даних технічних засобів (зокрема літаків), пов'язаних з технічним обслуговуванням, також є можливим застосуванням цієї технології завдяки незмінній хмарній книзі. Журнали технічного обслуговування, що зберігаються у хмарі, дозволяють технікам приймати обґрунтовані та точні рішення щодо заміни, ремонту та інших важливих завдань з технічного обслуговування, а власники можуть бути спокійними, знаючи про стан їх інвестиції.

1.3 Висновки по розділу

Завдяки багатьом його властивостям, блокчейн має можливість підвищити безпеку, прозорість, підзвітність та ефективність у багатьох організаціях та галузях. Ця технологія підвищує конфіденційність та зводить на нуль необхідність регулювання транзакцій третіми особами, що робить їх швидшими та більш надійними.

По цій причині блокчейн варто розглядати як те, чим він є – значною вертикальною інновацією, що змінює та продовжувати змінювати реалії багатьох індустрій. Також це є прямою причиною, з якої необхідне подальше вивчення їх потенціалу, у тому числі у їх основному застосуванні – криптовалютах. Саме тому у наступному розділі ми розглянемо основну тему цієї роботи, основні типи методів що забезпечують базовий рівень функціоналу блокчейну – алгоритми консенсусу, на прикладі їх застосування у криптовалютах.

РОЗДІЛ 2: Типи консенсусу в блокчейні, порівняльна характеристика і напрямки застосування

2.1 Proof-of-Work (PoW)

Proof-of-Work (англ. «доказ роботи»), часто скорочений до PoW, є найстаршим з типів консенсусу. Він полягає у створенні задачі яку складно розв'язати але легко перевірити. Рішення задачі є доказом проведеної роботи. Мета PoW – зробити так, щоб створення великої кількості нових дій з метою нашкодити системі (спам імейли, DoS-атаки) вимагало дуже великої кількості обчислюваних сил. У блокчейні PoW використовується для того щоб зробити переобрахування блоків для їх заміни настільки складним, що будь-який шахрай не зможе отримати достатню кількість обчислювальної сили для обходу правила «найдовшої ланки».

Bitcoin та всі криптовалюти, що походять від нього, використовують Hashcash PoW, розроблений у 1997 році Адамом Беком для запобігання імейл спаму. Принцип функціонування PoW наступний:

"Скажімо, базовий рядок, над яким ми будемо працювати, - це "Hello, World!". Наша мета полягає в тому, щоб знайти його варіант, який SHA-256 хешує до значення, меншого ніж 2^240. Ми змінюємо рядок, додаючи цифру під назвою nonce («number used once») до його кінця, і перевіряємо чи його хеш менше від цільового 2^240. Пошук відповідності для " Hello, World!" займає у нас 4251 спроб.

```
"Hello, world!0" =>
1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64 = 2^252.253458683
"Hello, world!1" =>
e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8 = 2^255.868431117
"Hello, world!2" =>
ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7 = 2^255.444730341
...
"Hello, world!4248" =>
6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdf65cc0b965 = 2^254.782233115
"Hello, world!4249" =>
c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26ebb9e6 = 2^255.585082774
"Hello, world!4250" =>
0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9 = 2^239.61238653
```

Оскільки 4250 обчислень хешу це досить мала кількість роботи для сучасного комп'ютеру, у біткоїні ціль (2^240 у попередньому прикладі) регулюється для того, щоб встановлення консенсусу для кожного блоку займало приблизно 10 хвилин"[5].

Процес знаходження розв'язку називається майнингом, та користувач який знайде рішення першим надсилає його іншим майнерам для перевірки. Коли усі майнери погоджуються на одному розв'язку встановлюється консенсус, перший майнер отримує кілька нових біткоїнів і транзакцію за блок та починається робота над наступним блоком.

Хоч знаходження є випадковим процесом, швидкі комп'ютери, що можуть здійснювати більше операцій за секунду, очевидно будуть знаходити рішення частіше, ніж їх конкуренти. Це породжує певну «гонку озброєнь» для майнингу, і з часом використання все швидших і швидших машин при сталому часу розв'язку блоку робить і без того досить енергетично неефективний процес надзвичайно затратним. Саме тому, незважаючи на його простоту та надійність існує та використовується велика кількість альтернатив PoW.

Щоб підсумувати, переваги PoW полягають у:

- Надійності.
- Простоті.

Недоліки PoW полягають у:

- Непотрібних витратах енергії.
- Майнери з більш швидким та/або спеціалізованим технічним забезпеченням мають значно вищий шанс отримання винагород за майнинг.
- Малій швидкості підтвердження транзакцій.

Визначні криптовалюти:

- Bitcoin, найперша та найбільш популярна криптовалюта у світі;
- Ethereum, друга по популярності криптовалюта, що відома завдяки своєму статусу піонера в сфері смарт-контрактів та доступній віртуальній машині (щоправда, у невизначеному майбутньому планується перехід до PoS).

2.2 Proof-of-Stake (PoS)

Proof-of-Stake (англ. «доказ вкладу») – перша з запропонованих альтернатив PoW. Оскільки видобуток монет за допомогою PoW є неефективним з точки зору витрат часу та енергії, використання PoW значно обмежує можливості операції блокчейну. PoS пропонує вирішити цю проблему за допомогою досягнення консенсусу методом обчислення сумарного вкладу користувачів (їх балансу у тій криптовалюті, для якої знаходиться консенсус), що пропонують певне рішення блоку. Як тільки цей вклад переважає 51%, консенсус вважається досягнутим. Винагорода ділиться поміж вкладниками відповідно до розміру їх вкладу.

Це означає, що чим більшим є баланс певного користувача у криптовалюті, тим більшим є його внесок у консенсус – користувач що має 1% загального балансу системи на момент обчислення буде мати 1% загального «голосу» [6]. Таким чином, мета PoS – зробити будь-які атаки не занадто складними для обчислення, як в PoW, а занадто дорогими та інвестиційно небезпечними (оскільки будь-яка атака призведе до девальвації монет, у тому числі тих, якими атакуючий має володіти для проведення атаки) для проведення.

Проблема з PoS полягає у тому, що будь-яка особа чи група осіб, що володіють монополією (сумарно більше 51% всіх токенів криптоактиву) можуть практично керувати функціонуванням блокчейну так, як вони забажають, та будуть завжди отримувати повний об’єм винагороди за підтвердження транзакції. Таким чином, формування монополії є фактично оптимальним рішенням для будь-якого користувача.

Переваги PoS:

- Низьке використання ресурсів.
- Надійність – перехоплення контролю над системою вимагає витрати що становлять щонайменше 51% капіталізації криптоактиву.
- Стабільність – навіть при встановленні монополії мотивація знищення системи є занадто малою щоб скористатись можливістю.

Недоліки:

- Оскільки консенсус вимагає абсолютної більшості, транзакції не мають сталого ліміту часу для підтвердження.
- Отримання винагород за високий баланс у криптоактиві мотивує не витрачати його, обмежуючи торгівлю.

Визначні криптовалюти:

- Dash – криптовалюта, побудована на основі ядра Bitcoin з додатковою конфіденційністю та функціоналом транзакцій, таким як PrivateSend та InstantSend.
- Binance Coin – криптовалюта за якою стоїть найбільша платформа обміну криптовалютами, Binance. Використовується для багатьох операцій на платформі.

2.3 Delegated Proof-of-Stake (DPoS)

Delegated Proof-of-Stake (англ. «делегований доказ вкладу») – це модифікація алгоритму PoS, що полягає у обмеженні кола майнерів шляхом вибору «делегатів» та «свідків». Ці ролі є непостійними та можуть мати різні вповноваження у різних криптовалютах, але загалом, обов'язки свідків полягають у перевірці транзакцій. В залежності від валюти свідки можуть блокувати проходження транзакцій, але оскільки блокування не лише злоякісних транзакцій негативно вплине на їх репутацію та ціну криптовалюти (а отже, їх баланс у ній), зловживання владою свідка відбувається рідко. Найбільш ефективні свідки отримують нагороди від системи у вигляді комісії за транзакції. [7]

Делегати здебільшого не мають можливості валідувати блоки, але частиною їх вповноважень є визначення властивостей блоків, таких як кількості транзакцій у них чи комісії за один блок. Делегати не отримують винагород від мережі, але властивості які вони визначають змінюються нечасто та вони можуть покращувати власну репутацію для того, щоб стати свідками у майбутньому.

Під час процесу вибору кожен користувач системи може голосувати за будь-якого кандидата в свідки чи делегати що бере участь у виборах. Їх голос пропорційний їх вкладу в криптовалюті (так само, як під час визначення консенсусу в PoS). Нижній ліміт балансу для голосування відсутній.

Переваги DPoS над PoS:

- Швидкість – транзакції перевіряються свідками практично миттєво, та за відсутності одного свідка його транзакції автоматично передаються іншим. DPoS швидший, ніж PoW та PoS.
- Демократичність – система дозволяє більшості користувачів обирати зміни до функціоналу криптовалюти за допомогою делегатів.

Недоліки у порівнянні з PoS:

- Існування криптовалюти повністю залежить від ентузіазму користувачів.
- Знижений рівень децентралізації.

Визначні криптовалюти:

- EOS – це блокчейн-протокол, який дозволяє горизонтальне масштабування децентралізованих програм, що дозволяє розробникам ефективно створювати високопродуктивні розподілені програми.
- Tezos – демократична криптовалюта, що підтримує формальну перевірку – методику, яка математично підтверджує правильність коду, що регулює транзакції, та використовує підвищені стандарти безпеки.

2.4 Leased Proof-of-Stake (LPoS)

Leased Proof-of-Stake (англ. «орендований доказ вкладу») – це модифікація стандартного алгоритму PoS, що імплементована в криптовалюті Waves. Процес досягнення консенсусу є схожим до PoS, але з невеликою зміною – транзакції валідуються випадково обраним колом користувачів. Вони обирається з списку повних нод системи, а вірогідність вибору залежить від балансу користувача, включаючи орендовані йому токени [8].

LPoS дозволяє користувачам які з тих чи інших причин не хочуть чи не можуть брати участь у майнингу «орендувати» власні кошти іншим користувачам. У якості винагороди вони можуть отримувати відсотки комісії за транзакції, що були перевірені користувачем котрому вони орендували свої кошти.

Переваги LPoS над PoS:

- Безпека – повні ноди, які підключені до Інтернету, не мають повного доступу до усіх токенів що використовуються для консенсусу, роблячи хакерські атаки на них менш ефективними.
- Доступність – система орендування дозволяє навіть користувачам, що не мають можливості запускати повні ноди, підтримувати екосистему криптовалюти.

Недоліки:

- Завдяки можливості отримання нагород без необхідності витрачати гроші на підтримання майнингового обладнання, існує стимул не запускати повні ноди, що зменшує стабільність системи.
- Знижений рівень децентралізації.

Поки що єдиною криптовалютою що підтримує цей тип консенсусу є Waves – децентралізована платформа з акцентом на доступність, безпеку та простоту інтеграції.

2.5 Proof-of-Authority (PoA)

Proof-of-Authority (англ. «доказ авторитету») – це метод консенсусу, що полягає у перевірці та затвердженні блоків лише «авторитетними» користувачами. «Авторитет» визначається за допомогою вбудованої системи репутації, що є різними для різних систем. Коло користувачів з правом затвердження блоків обирається заздалегідь згідно з визначеними стандартами. [9]

Лише користувачі з високою репутацію можуть атакувати систему, після чого вони втратять своє право на участь в функціонуванні блокчейну. Цей метод консенсусу, зі зрозумілих причин, не використовується в публічних блокчейнах.

Переваги PoA:

- Висока швидкість створення блоків.
- Високий рівень контролю учасниками блокчейну, що робить PoA привабливим для корпорацій.

Недоліки PoA:

- Відсутність анонімності.
- Можливість встановлення цензури.

На момент написання цієї роботи існує лише дві криптовалюти, що використовує PoA – CFX та POA. Обидва проекти є досить новими та маловідомими, тому щось сказати про їх успіх та фокус складно.

2.6 Proof-of-Importance (PoI)

Proof-of-Importance (англ. «доказ важливості») – це метод консенсусу, що використовується у криптовалюті NEM. Розробники NEM називають його «одним з головних нововведень NEM у галузі блокчейн» та кажуть що «це новий алгоритм, який використовує теорію мережі для визначення рейтингу важливості кожного облікового запису в мережі» [10].

Він у багатому схожий на LPoS, оскільки дозволяє орендувати свої ресурси основній ноді для отримання винагород за затвердження транзакції без необхідності використання власного обладнання для проведення обчислень, проте відрізняється тим, що замість власного вкладу в криптовалюту користувач має орендувати свою «важливість».

Важливість вираховується з багатьох факторів, включаючи (але не обмежуючись): кількість та розміром транзакцій загалом та за останній місяць, баланс користувача в криптовалюті та важливість користувачів, з якими торгував даний користувачів [10].

Переваги у порівнянні з LPoS:

- Активність – система важливості створює стимул до обміну з іншими користувачами.

Недоліки:

- Низький рівень безпеки – завдяки алгоритму системи важливості існує небезпека використання програмного забезпечення для штучного збільшення рівня важливості з метою захоплення системи.

Система PoI на даний момент використовується лише криптовалютою NEM – платформа, що слугує для створення швидких та стабільних систем для компаній різного розміру.

2.7 Proof-of-Capacity (PoC)

Proof-of-Capacity (англ. «доказ розміру») – це метод консенсусу, що дозволяє майнерам виконувати затвердження транзакцій за рахунок вільного місця на

жорсткому диску їх комп'ютера. Він дуже схожий на Proof-of-Work, оскільки шанс отримання винагороди за транзакцію залежить від характеристик технічного забезпечення, а не від певної позиції користувача у блокчейні.

"Затвердження блоку проходить у два етапи: етап планування та етап майнингу.

У першому етапі проводиться планування – створюються 8192 хешів для кожного можливого нонсу, кількість нонсів обмежена лише вільною пам'яттю на жорсткому диску комп'ютеру. Ці хеші поєднуються у пари, утворюючи 4096 пар – «скупів».

У другому етапі проходить власне обчислення умов майнингу та сам процес затвердження блоку. Наприклад, скажімо, що комп'ютер генерує скуп № 38. Майнер перевірить скуп № 38 першого нонсу, отримає з нього певний дедлайн, після чого перевірить скуп №38 другого нонсу, і так далі. Після перевірки усіх значень комп'ютер обирає найменший з дедлайнів.

Цей дедлайн представляє тривалість часу в секундах, що має пройти з моменту останнього підтвердженого цим комп'ютером блоку до того, як йому буде дозволено підтвердити новий блок. Якщо блок не буде підтвердженим до того, як сплине дедлайн, комп'ютер може підтвердити цей блок та отримати винагороду за нього".[11]

Переваги у порівнянні з PoW:

- Висока швидкість.
- Значно нижче використання ресурсів.

Недоліки:

- Процес майнингу може бути зупиненим зловмисним програмним забезпеченням.
- Нижча ціна виконання атаки 51% на систему.

2.8 Byzantine Fault Tolerance

Byzantine fault (англ. «візантійська похибка») – це проблема комп'ютерних систем у яких інформація, що надходить від певних учасників системи, є ненадійною. Ця проблема особливо сильно стосується децентралізованих систем, де покази кожного з учасників є рівносильними.

Коли поламаний компонент починає надсилати різні сигнали різним частинам системи, стає необхідністю виявлення того, який саме з компонентів є поламаним (оскільки як відправник, так і адресат можуть надавати недостовірні дані).
Назва походить від знаменитої проблеми візантійських генералів, описаної у 1982 році Лампортом, Шостаком та Пісом, яка, у скороченому формулюванні, звучить так: «Кожен з генералів має надіслати накази своїм лейтенантам так, що:

- Усі лояльні лейтенанти виконають один і той самий наказ.
- Якщо генерал лояльний, усі лояльні лейтенанти мають виконати його наказ» [12].

Візантійські похибки вважаються найбільш загальним і найскладнішим класом відмов серед режимів відмов. Вони не передбачають ніяких обмежень, а це означає, що невдалий вузол може генерувати довільні дані, включаючи дані, які роблять його схожим на функціонуючий вузол. Таким чином, візантійські похибки можуть сплутати системи виявлення несправностей, що ускладнює стійкість до відмов. Незважаючи на аналогію, візантійська невдача не обов'язково є проблемою безпеки, що передбачає вороже втручання людини: вона може виникати виключно з електричних чи програмних несправностей. [13]
Оскільки блокчейни були задумані як децентралізовані системи, у них неможливо організувати сертифікати довіри, що робить їх значно менш стійкими до візантійських похибок. Без BFT кожен користувач зміг би порушити стабільність системи шляхом посилання конфліктуючих транзакцій. Для існування блокчейну повинні бути забезпечені дві вищеописані умови – усі доброчесні користувачі виконають один наказ згідно правилу найдовшого ланцюга (створення ланцюга довшого за ланцюг доброчесних користувачів вимагає більшу кількість необхідних ресурсів у шахраїв, що є теоретично неможливим для достатньо великої мережі), та усі доброчесні користувачі завірять доброчесні транзакції (оскільки кількість доброчесних користувачів теоретично вища за кількість зрадників).

2.9 Directed Acyclic Graph (DAG)

Directed Acyclic Graph, або спрямований ациклічний граф – це основа для методів консенсусу яка принципово відрізняється від блокчейну та методів консенсусу для

нього, таких як Proof-of-Work чи Proof-of-Stake.

“Якщо спростити блокчейни до базових структур даних, їх можна вважати простими пов'язаними списками. Кожний новий запис у Bitcoin або Ethereum (або в інших мережах) ставиться після попереднього і посилається на нього. В результаті отримується лінійна послідовність блоків, яку ми називаємо ланцюгом.

Блокчейни дозволяють відстежувати будь-які записи, збережені в історії леджеру, але їх послідовна структура також є найбільшим обмеженням пропускної здатності транзакцій у них; природа блокчейну як плоского списком є створює значні проблеми для його здатності до масштабування.

Що ж, DAG діє по-іншому. Ця структура даних нагадує блок-схему, де всі точки спрямовані в одному напрямку. Ви можете порівняти спрямований ациклічний графік (DAG) зі структурою файлових файлів, де у папках є підпапки, які розгалужуються на інші підпапки тощо; вони деревоподібні.”[14]

В даному випадку, спрямованість означає, що усі зв’язки у графі мають однозначно визначений напрямок; ациклічність – що не може існувати випадку, у якому нода графу буде посилатись сама на себе або на ноди, що прямим чи непрямим чином посилаються на неї.

Основною платформою, що використовує DAG, є IOTA. У їх DAG, Tangle, кожна нова транзакція повинна ухвалити щонайменше дві попередні транзакції для того, щоб її можна було ухвалити іншим нодам.

Між нодами графу використовуються інші алгоритми консенсусу, зокрема PoW чи PoC, але оскільки об’єм графу значно збільшує кількість потужності, необхідну для перехоплення мережі, складність задачі вирахування хешу, а отже і кількість роботи чи пам’яті, необхідної для цього, значно зменшується у порівнянні з стандартними варіантами цих типів консенсусу.

2.10 Висновки по розділу

Існує досить велика кількість різних типів консенсусу, кожен з яких має свої переваги та недоліки. Вони усі намагаються розв’язати одну проблему – проблему підтвердження інформації кожного з користувачів згідно з загальними правилами, мінімізуючи можливі негативні наслідки від дій шахраїв та можливих технічних

проблем системи, але кожен з них пропонує дуже різні методи вирішення цієї проблеми. Завдяки цьому вони мають різні пріоритети розробки та сфери пристосування, та базове знання усіх типів консенсусу необхідне для розробки блокчейн-проектів.

Загалом, пошук нових, більш досконалих алгоритмів консенсусу триває щодня, і кожна з цих нових знахідок має шанс змінити увесь світ комп'ютерних технологій на краще.

РОЗДІЛ 3: Конкретні приклади і перспективи застосування технології блокчейн в Україні

3.1 Статус технологій

На момент написання курсової роботи існує досить мала кількість інформації про активні використання технології в Україні. Незважаючи на те, що в Україні працює значна кількість блокчейн-компаній (482.solutions, Bloqly, Attic Lab, тощо) та укладеному у квітні 2017 року контракту між BitFury та Україною [15], деякі з анонсованих технологій для публічного сектору є недоступними з невідомих причин на момент написання.

3.2 Bloqly

Якщо вірити вебсайту цієї компанії, Bloqly вносить значний внесок у створення нових блокчейн-орієнтованих технологій у публічному секторі України. Вони зазначають такі проекти:

- Проект з розробки автоматизованої системи видачі автосертифікатів разом з Міністерством інфраструктури України.
- Системи захисту банківських архівів, банківських гарантій та автоматизованої системи взаємодії з ліцензіатами разом з НБУ.
- Проекти з системи супроводу вантажів на блокчейні та реєстрів судоходства разом з Державною службою морського та річкового транспорту.
- Проекти Smart City разом з міськими адміністраціями Львова та Дрогобича. [16]

Деталі функціонування їх блокчейн рішення невідомі, але, якщо вірити документації у GitHub, блокчейн Bloqly використовує PoA консенсус. У даному випадку цей вибір легко зрозуміти – PoA працює найкраще для логістичних чи документаційних цілей у приватних мережах, що є якраз тим, чим займається дана компанія. Проте достовірність цієї інформації перевірити неможливо.

3.3 Система електронних торгів арештованим майном (СЕТАМ)

8 вересня 2017 року був анонсований факт переходу ДП «СЕТАМ» на блокчейн та його ребрендинг в OpenMarket [17].

Оновлений аукціон працює на базі фреймворку Exonum, розробленому компанією BitFury. Цей фреймворк працює з допомогою BFT консенсусу, що походить від системи консенсусу Tendermint, але «має кілька відмінних характеристик порівняно з ним та іншими алгоритмами консенсусу для блокчейнів» [18].

На даний момент блокчейн використовується не у всіх частинах OpenMarket, триває робота по повному перенесенні системи на блокчейн. На даний момент працює зберігання ставок у блокчейні – їх статус можна перевірити за допомогою хешів блоку ставки.

3.4 Технології з невідомим статусом

Вищезгадані технології є не єдиними з анонсованих. Наступні технології були анонсованими або створеними, але зараз знаходяться в невідомому статусі (наприклад, домен сервісу не зареєстрований):

- E-Vox: блокчейн-платформа для забезпечення демократичних онлайн-виборів.
- E-Auction 3.0: блокчейн-платформа для забезпечення відкритих, безпечних та швидких онлайн-аукціонів.
- E-Hryvnya: електронний стейблкоїн прив'язаний до гривні, зроблений з метою створення електронного аналогу гривні. На даний момент знаходиться в стані невизначеності, оскільки централізована природа необхідна для функціонування такого проекту робить використання блокчейну менш корисним. [19]

3.5 Висновки до розділу

Україна є дуже перспективним полем для створення нових блокчейн-проектів, але інформацію про успішне їх використання неймовірно складно знайти.

Варто підмітити, що блокчейн-технологія є дуже корисною для України, враховуючи надзвичайно високий рівнем корупції та засилля неефективної бюрократії у нашому державному апараті, оскільки завдяки своїй надійності та неможливості непомітної зміни даних вона значно підвищує рівень довіри та унеможливлює певні аспекти корупції. Це, звичайно, далеко не єдина перевага можливого використання блокчейну в публічному секторі, але це, на думку автора, робить видиме припинення підтримки технологій, що описані вище як ті, що знаходяться в невідомому стані, абсолютно незрозумілим.

Тим не менш, очевидно що розробка та інтеграція нових систем йде повним ходом, та завдяки відсутності інформації можливо, що ступінь використання даних технологій є значно вищим, ніж описано в даній роботі.

РОЗДІЛ 4: Практична робота

4.1 Коротке формулювання завдання

Завдання «Журнал трейдера» полягає у створенні мобільного додатку для спрощення задачі трейдингу.

Мобільний додаток дозволить трейдерам криптовалют вести журнал трейдера у зручній формі шляхом запису всіх своїх торгових операцій і корегувати свою власну торгову стратегію для поліпшення результатів торгівлі. Цей журнал дозволить заносити інформацію про свої торгові операції, суму депозита, портфель криптовалют, аналізувати результати торгівлі і корегувати власну стратегію.

4.2 Обґрунтування вибору середовища

У якості середовища був обраний фреймворк Django для роботи з БД, СУБД SQLite та HTML+CSS+Python для створення інтерфейсу.

Django був обраний завдяки:

- Повноті – Django включає в себе більшість функцій, необхідних для виконання завдання. Той факт, що усі необхідні частини є частиною одного фреймворку забезпечує просту і швидку роботу додатку;
- Безпеці – оскільки для функціонування додатку потрібні деякі маніпуляції з особистими даними користувачів, необхідно забезпечити певний рівень безпеки. Django надає значну кількість засобів для автоматичного захисту веб-сайту. Django забезпечує захист від багатьох вразливостей за замовчуванням, включаючи ін'єкцію SQL, кроссайтове скриптування, підробку запитів з інших веб-сайтів, тощо;
- Компактності – Django дозволяє писати код з використанням принципів дизайну та зразків, що заохочує багаторазове використання коду. Завдяки принципу DRY (“don’t repeat yourself”, або "не повторюй себе"), зменшена кількість дубльованого коду. Django також сприяє групуванню пов'язаних функціональних можливостей у багаторазові "програми", а на нижчому рівні групує пов'язаний код у модулі;

Вибір HTML+CSS+Python був зроблений як наслідок вибору Django.

SQLite був обраний завдяки:

- Простоті – динамічна природа SQLite значно спрощує процес роботи з БД;

- Компактності – SQLite не вимагає додаткових процесів або певних залежностей для функціонування;
- Надійності – усі транзакції в SQLite дотримуються стандарту ACID. Це означає, що всі запити та зміни є атомними, послідовними, ізольованими та довговічними. Іншими словами, всі зміни в межах транзакції відбуваються повністю або зовсім не відбуваються, навіть коли трапляється несподівана ситуація, наприклад, збої програми, відключення живлення або збої операційної системи.

Для виконання завдання був необхідний вибір API для отримання даних про ціни криптоактивів. У зв'язку з відсутністю прив'язки до однієї конкретної біржі та великої кількості інформації був обраний CoinApi.

4.3 Принципи реалізації функцій системи

Для правильної роботи системи необхідний певний набір первинних функцій, які зазначені у Додатку Г.

Принцип реалізації цих функцій полягає у наступному:

- відображення суми депозиту у фіатних грошах відбувається завдяки створенні у БД моделі Активу:

```
class Active(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    currency = models.CharField(max_length=5)
    amount = models.DecimalField(max_digits=16,decimal_places=8)
    is_initial = models.BooleanField(default=False)
```

- Та реалізації зв'язку з зовнішнім API для отримання актуальної інформації про поточну ціну криптоактиву:

```
def get_from_api(query):
        url = API_URL+query
        headers = {'X-CoinAPI-Key' : API_KEY}
        response = requests.get(url, headers=headers)
        return response.json()
```

```
def get_in_usd(self):
        if self.currency=='USD':
            return self.amount
        query = f'/exchangerate/{self.currency}/USD'
        return conv.get_from_api(query).get('rate')*self.amount
```

- Занесення дати операції, суми купівлі/продажу, відсотку та типу комісії відбувається завдяки моделі Операції:

```
class Operation(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    datetime = models.DateTimeField(default=timezone.now,verbose_name='Date and time at wh
ich the operation was closed',null=True)
    currency_bought = models.CharField(max_length=20)
    currency_sold = models.CharField(max_length=20)
    exchange_name = models.CharField(max_length=30)
    amount_bought = models.DecimalField(max_digits=16,decimal_places=8)
    commission_percentage = models.DecimalField(max_digits=5,decimal_places=3)
    buy_rate = models.DecimalField(max_digits=9,decimal_places=4)
    eventual_rate = models.DecimalField(max_digits=9,decimal_places=4,default=0)
    is_maker = models.BooleanField()
    is_open = models.BooleanField(default=True)
```

- Автоматичне встановлення кінцевої ціни (ціни, до якої падав чи піднімався криптоактив) відбувається завдяки методу get_eventual_rate у класі Operation. Перевірка факту падіння чи підняття ціни криптоактиву відбувається кожні 12 годин. Процес перевірки виглядає так:

```
def get_eventual_rate(self):
        if self.is_open == True:
            raise ValueError('The transaction is still open')
        first_pass = True
        prev_rate = 0
        check_time = self.datetime
        while True:
            check_time += datetime.timedelta(hours=12)
            if check_time<datetime.now():
                rate = self.get_rate(check_time)
                if first_pass:
                    prev_rate = rate
                    first_pass = False
                    continue
                else:
                    if prev_rate<self.buy_rate:
                        if rate>prev_rate:
                            self.eventual_rate=prev_rate
                            break
                    else:
                        if rate<prev_rate:
                            self.eventual_rate=prev_rate
                            break
                prev_rate = rate
            else:
                p = Process(target=self.__wait_12_hours)
                p.start()
        self.save()
```

- Отримання прибутку операції відбувається за двома різними принципами. Прибуток у відсотках визначається з існуючих у БД даних. Прибуток у $ визначається за допомогою API:

```
def get_profit(self):
        profit_in_cur = (self.eventual_rate-self.buy_rate)*self.amount_bought
        query = f'/exchangerate/{self.currency}/USD'
        return conv.get_from_api(query).get('rate')*profit_in_cur
```

4.4 Взаємодія користувачів з системою

У системі номінально існує лише один тип користувачів (також існує особлива сутність – адміністратор системи, але його роль у функціонуванні системи не обов'язково потребує прямої взаємодії і, як наслідок, сутність адміністратора може бути видаленою в будь-який момент без впливу на стабільність системи).
У будь-якого авторизованого користувача є такі функції, за допомогою яких вони можуть працювати з даними, збереженими у системі:

- Функція додавання до власного акаунту будь-якого з існуючих у API активів
  - Актив вважається існуючим у API, якщо він наявний у списку, що можна отримати за допомогою запиту до https://api.coinapi.io/v1/assets/
- Можливість перегляду стану активів, що під'єднані до їх акаунту
- Можливість прямої зміни кількості певного активу, що належить користувачу
- Можливість видалення активів з БД
- Можливість додавання операцій до власного акаунту, якщо вони не порушують цих продукційних правил:
  - Операції не можуть мати майбутній час. Це зумовлюється тим, що будь-яка операція додана до системи є лише відображенням операції, що вже сталась на певній криптовалютній біржі.
  - Операції не можуть оперувати активами, що не існують у API, оскільки це не дозволяє системі відстежувати їх ціну.
  - Операції не можуть працювати з недодатними кількостями активів, оскільки неможливо купити нуль або від'ємну кількість токенів
  - Операції не можуть мати від'ємний відсоток комісії, оскільки це суперечить функціоналу біржі криптовалют.

- Співвідношення купленого активу до проданого (позначене як buy_rate у БД) не може бути від'ємним або нульовим, оскільки таке відношення означатиме від'ємну або нульову ціну одного з активів, що не є можливим.

Примітка – операції, що перевищують ліміт певного звітного періоду, можуть бути доданими, оскільки вони вже стались та заборона їх додавати не несе сенсу з аналітичної точки зору, але період буде помічений як такий, для якого перевищений один з визначених лімітів до зміни ліміту або видалення однієї з операцій.

- Можливість переглядати свої попередньо додані операції.
- Можливість редагувати свої операції, якщо результуючі операції не порушують правила, зазначені вище.
- Можливість видаляти свої операції.
- Можливість визначення для свого акаунту звітних періодів, що не порушують цих продукційних правил:
  - Періоди не можуть перетинатись по часу. Будь-якому моменту в часі має відповідати один або нуль періодів, оскільки в інакшому випадку не може бути гарантованою коректна робота деяких функцій системи, зокрема функцій сигналізації про перевищення ліміту.
  - Максимальний розмір портфелю криптоактивів за період має бути не меншим за 2, оскільки початковий депозит також вважається активом, а отже встановлення ліміту на 1 чи менше буде автоматично порушувати ліміт. Також цей розмір не може перевищувати 32768 у зв'язку з обмеженнями бази даних.
  - Вікно часу, на який діє ліміт кількості операції, обирається з наступних трьох значень: "Day" (1 день), "Week" (1 тиждень) та "Month" (1 місяць).
- Можливість редагувати свої періоди, якщо результуючі операції не порушують правила, зазначені вище.
- Можливість видаляти свої періоди.

- Можливість редагувати дані власного акаунту, зокрема назву, пароль та електронну адресу.

Для забезпечення взаємодії користувачів з системою використовуються HTML файли з додатковим використанням мови шаблонів Django. Ця мова дозволяє відображення інформації з бази даних напряму. Наприклад, базова частина сайту, яка використовується у кожній сторінці додатку, виглядає так:

```
<!DOCTYPE html>
<html lang="en">
<head>
  {% block title %}<title>Trade Journal</title>{% endblock %}
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/boot
strap.min.css" integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
  {% load static %}
  <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>
<body>
  <div class="container-fluid">
    <div class="row">
    {% if messages %}
    <ul class="messages">
        {% for message in messages %}
        <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>
        {% endfor %}
    </ul>
    {% endif %}
      <div class="col-sm-2">
      {% block sidebar %}
        <ul class="sidebar-nav">
          <li><a href="{% url 'trader_journal:index' %}">Home</a></li>
          {% if user.is_authenticated %}
          <li>User: {{ user.get_username }}</li>
          <li><a href="{% url 'trader_journal:profile' %}?next={{request.path}}">Profile</
a></li>
          <li><a href="{% url 'trader_journal:logout' %}?next={{request.path}}">Logout</a>
</li>
          <li><a href="{% url 'trader_journal:actives' %}">Your actives</a></li>
          <li><a href="{% url 'trader_journal:operations' %}">Your operations</a></li>
          <li><a href="{% url 'trader_journal:periods' %}">Your periods</a></li>
          {% else %}
          <li><a href="{% url 'trader_journal:login' %}?next={{request.path}}">Login</a></
li>
          <li><a href="{% url 'trader_journal:register' %}?next={{request.path}}">Register
</a></li>
          {% endif %}
```

```
        </ul>
    {% endblock %}
     </div>
     <div class="col-sm-10 ">{% block content %}{% endblock %}</div>
   </div>
  </div>
</body>
</html>
```

За допомогою цього інструменту створюються динамічні посилання, відображається інформація про користувача, інформаційні повідомлення та відбувається загруження статичного контенту.

4.5 Структура інтерфейсу системи

Система включає в себе певну кількість веб-сторінок, що є інструментом для взаємодії між користувачем та «зовнішнім» рівнем системи.

Перелік цих сторінок та їх функціоналу включає:

- Основну сторінку, або індекс. На ній є можливість:
    - Переглянути кількість доданих користувачем операцій.
    - Дізнатись про поточний період.
    - Переглянути прибуток поточного періоду.
    - Переглянути прибуток з початку трейдингу.
    - Дізнатись про загальний баланс акаунту.
    - Дізнатись про баланс активів, що позначені як початковий депозит.
    - Переглянути останні операції акаунту.
    - Переглянути список найбільш значних активів акаунту.
- Сторінку регістрації. На ній є можливість:
    - Створити нового користувача.
- Сторінку профіля. На ній є можливість:
    - Переглянути інформацію про акаунт: електронну адресу та ім’я користувача.
    - Редагувати зазначену інформацію про акаунт.
    - Перейти на сторінку зміни паролю.
- Сторінку списку активів/періодів/операцій. На ній є можливість:
    - Переглянути список активів/періодів/операцій цього акаунту.

    - Видалити будь-який з активів/періодів/операцій.
    - Перейти до сторінки конкретного активу/періоду/операції.
- Сторінку активу/періоду/операції. На ній є можливість:
    - Переглянути усю інформацію про актив/період/операцію.
    - Перейти на сторінку редагування операції.
    - Видалити даний актив/період/операцію.

4.6 Висновки до розділу

Було розроблено додаток «Журнал трейдера», що надає підвищений комфорт при використанні криптовалютних бірж для трейдингу. Під час виконання завдання були вивчені особливості трейдингу криптовалют, розроблені алгоритми зв’язку з стороннім API та інтеграції криптовалютних термінів до стандартних баз даних. Можливий подальший розвиток продукту для підвищення комфорту користування шляхом удосконалення інтерфейсу та отримання прямого доступу до активів та операцій користувача за допомогою API певних бірж.

ВИСНОВКИ

У даній роботі був розглянутий блокчейн в цілому та типи консенсусу зокрема, включаючи принципи їх функціонування, їх переваги та недоліки, та їх потенційні та наявні застосування.

Методи консенсусу є однією з базових технологій, що забезпечує функціонування блокчейну, оскільки вони вирішують одну з головних проблем розподілених систем – проблему встановлення загального рішення незважаючи на можливі похибки системи, згідно з базовими правилами системи. Кожен з методів, розглянутих у цій роботі, здатен вирішити цю проблему, зазвичай за допомогою правила найдовшого ланцюга, проте те, як саме встановлюється рішення, значно відрізняється між ними. Наприклад, в той час як PoW вимагає значної кількості обчислень для того, щоб зробити «хибні» операції незначними, PoA практично не вимагає обчислень для встановлення консенсусу, але вимагає значної кількості інформації про користувачів для цієї ж мети. Завдяки цій фундаментальній різниці усі методи консенсусу мають дуже різні сфери використання та власні переваги та недоліки. Ці характеристики були аналізовані та описані у цій роботі.

Для аналізу можливостей допрацювання цієї роботи, варто зазначити, оскільки методи консенсусу у певній мірі відрізняються майже в кожному використанні блокчейн-технології, для створення конкретних застосувань є доцільним детальне вивчення алгоритмів функціонування існуючих використань методів, що не було зроблено у цій роботі. Також можна провести ці дослідження знову у майбутньому, коли будуть створені нові методи консенсусу, оскільки середовище блокчейн-проектів є надзвичайно нестабільним.

Загалом, блокчейн є інноваційною технологією, а типи консенсусу є частиною фундаменту його існування, тому винайдення нових типів консенсусу може бути значним рушієм прогресу для багатьох індустрій одночасно. Подальше вивчення цієї теми неминуче буде корисним.

Список використаної літератури

1. Blockchain [Електронний ресурс] : Матеріал з Вікіпедії — вільної енциклопедії : Версія 954984370, збережена о 09:47 UTC 5 травня 2020 / Автори Вікіпедії // Вікіпедія, вільна енциклопедія. — Електрон. дан. — Сан-Франциско: Фонд Вікімедіа, 2016. — Режим доступу: https://en.wikipedia.org/wiki/Blockchain
2. Why Bitcoin Matters [Електронний ресурс] : стан на 21 травня 2020 / Marc Andreessen // Режим доступу: https://genius.com/Marc-andreessen-why-bitcoin-matters-annotated
3. A dozen companies that reaped rewards by putting “bitcoin” or “blockchain” in their name [Електронний ресурс] : стаття від 12 січня 2018 / John Detrixhe // Режим доступу: https://qz.com/1175701/putting-bitcoin-or-blockchain-in-a-company-name-is-sometimes-enough-for-a-pop-on-the-stock-market/
4. Bank mit Kette [Електронний ресурс] : стаття від 18:50 11 грудня 2018 // Режим доступу: https://www.sueddeutsche.de/wirtschaft/blockchain-bank-mit-kette-1.4248410
5. Proof of Work [Електронний ресурс] : Матеріал з Bitcoin Wiki: Версія 66393, збережена о 00:41 UTC 24 квітня 2020 // Режим доступу: https://en.bitcoin.it/wiki/Proof_of_work
6. Proof of Stake [Електронний ресурс] : Матеріал з Bitcoin Wiki: Версія 66738, збережена о 13:34 UTC 20 вересня 2019 // Режим доступу: https://en.bitcoin.it/wiki/Proof_of_Stake
7. Delegated Proof of Stake [Електронний ресурс] : Матеріал з BitcoinWiki: Версія 383806, збережена о 10:50 UTC 3 травня 2020 // Режим доступу: https://en.bitcoinwiki.org/wiki/DPoS
8. Leased Proof of Stake [Електронний ресурс] : офіційна документація WAVES, стан на 21 травня 2020 // Режим доступу: https://docs.wavesprotocol.org/en/blockchain/leasing
9. Proof of Authority Explained [Електронний ресурс] : стан на 21 травня 2020 // Режим доступу: https://www.binance.vision/blockchain/proof-of-authority-explained
10. Proof of Importance (POI) [Електронний ресурс] : офіційний сайт NEM, стан на 21 травня 2020 // Режим доступу: https://nem.io/xem/harvesting-and-poi/#proof-of-importance
11. Proof of Capacity (Cryptocurrency) [Електронний ресурс] : стан на 21 травня 2020 // Режим доступу: https://www.investopedia.com/terms/p/proof-capacity-cryptocurrency.asp
12. Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" [Електронний ресурс] / Leslie Lamport, Robert Shostak, Marshall Pease // ACM Transactions on Programming Languages and Systems. – Режим доступу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.9525&rep=rep1&type=pdf
13. Byzantine fault [Електронний ресурс] : Матеріал з Вікіпедії — вільної енциклопедії : Версія 952447358, збережена о 08:21 UTC 22 квітня 2020 / Автори Вікіпедії // Вікіпедія, вільна енциклопедія. — Електрон. дан. — Сан-Франциско: Фонд Вікімедіа, 2016. — Режим доступу: https://en.wikipedia.org/wiki/Byzantine_fault
14. Cryptocurrencies Without Blockchain? Learn More About Directed Acyclic Graphs (DAG) [Електронний ресурс] : стан на 21 травня 2020 / Ivan Kohut // Режим доступу: https://perfectial.com/blog/dag-vs-blockchain/
15. Ukraine launches big blockchain deal with tech firm Bitfury [Електронний ресурс] : стаття від 9:35 17 квітня 2017 / Gertrude Chavez-Dreyfuss // Режим доступу: https://www.reuters.com/article/us-ukraine-bitfury-blockchain-idUSKBN17F0N2
16. Офіційний сайт Bloqly [Електронний ресурс]: стан на 21 травня 2020 // Режим доступу: https://www.bloqly.com
17. СЕТАМ став першим у світі аукціоном на blockchain та змінив назву на OpenMarket [Електронний ресурс] : офіційний сайт ДП «СЕТАМ» // Режим доступу: http://setam.gov.ua/article/setam-stav-pershim-u-sviti-auktsionom-na-blockchain-ta-zminiv-nazvu-na-openmarket
18. Analytical Report on the E-Hryvnya Pilot Project [Електронний ресурс]: Національний Банк України, 2019 // General Conclusions – ст. 33 п. 7 – Режим доступу: https://bank.gov.ua/admin_uploads/article/Analytical%20Report%20on%20E-hryvnia.pdf
19. Consensus in Exonum [Електронний ресурс] : офіційна документація Exonum, версія 0.12 // Режим доступу : https://exonum.com/doc/version/0.12/architecture/consensus/

Додаток А. Лістинг коду.

Models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
from datetime import date
from django.urls import reverse
from django.core.exceptions import ObjectDoesNotExist

import datetime
import time
from multiprocessing import Process

from . import conv


def get_current_period(user):
        try:
                current_period=Period.objects.get(user_id=user,date_end__gte=date.today(),
date_start__lte=date.today())
                return current_period
        except ObjectDoesNotExist:
                return None

class Active(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    currency = models.CharField(max_length=5)
    amount = models.DecimalField(max_digits=16,decimal_places=8)
    is_initial = models.BooleanField(default=False)

    def get_absolute_url(self):
        return reverse("trader_journal:active_detail", kwargs={"pk": self.pk})

    def get_in_usd(self):
        if self.currency=='USD':
            return self.amount
        query = f'/exchangerate/{self.currency}/USD'
        return conv.get_from_api(query).get('rate')*float(self.amount)

    def __str__(self):
        return str(self.currency)+': '+str(self.amount)

class Operation(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    datetime = models.DateTimeField(default=timezone.now,verbose_name='Date and time at wh
ich the operation was closed',null=True)
    currency_bought = models.CharField(max_length=20)
    currency_sold = models.CharField(max_length=20)
    exchange_name = models.CharField(max_length=30)
    amount_bought = models.DecimalField(max_digits=16,decimal_places=8)
```

```
    amount_sold = models.DecimalField(max_digits=16,decimal_places=8)
    commission_percentage = models.DecimalField(max_digits=5,decimal_places=3)
    eventual_rate = models.DecimalField(max_digits=9,decimal_places=4,default=None,null=Tr
ue)
    is_maker = models.BooleanField()
    is_open = models.BooleanField()

    def buy_rate(self):
        return self.amount_sold/self.amount_bought

    def get_absolute_url(self):
        return reverse("trader_journal:operation_detail", kwargs={"pk": self.pk})

    def get_profit(self):
        if self.eventual_rate is None:
            return 0
        return (self.eventual_rate-self.buy_rate())*self.amount_bought

    def get_profit_perc(self):
        if self.eventual_rate is None:
            return 0
        return (1-self.eventual_rate/self.buy_rate())*100

    def get_eventual_rate(self):
        if self.is_open == True:
            return
        first_pass = True
        prev_rate = 0
        check_time = self.datetime
        while check_time<timezone.now():
            check_time += datetime.timedelta(hours=12)
            rate = self.get_rate(check_time)
            if first_pass:
                prev_rate = rate
                first_pass = False
                continue
            else:
                if prev_rate<self.buy_rate():
                    if rate/prev_rate>=1.02:
                        self.eventual_rate=prev_rate
                        break
                else:
                    if rate/prev_rate<=0.98:
                        self.eventual_rate=prev_rate
                        break
                prev_rate = rate
        self.save()

    def get_period(self):
        return Period.objects.filter(date_start__lte=self.datetime,date_end__gte=self.date
time).order_by('date_end').first()
```

```
    def get_rate(self, datetime=datetime):
        if datetime is None:
            return None
        query = f'/exchangerate/{self.currency_bought}/{self.currency_sold}'
        query+= f'/?time={datetime.isoformat(timespec="microseconds")[:-6]}0Z'
        return conv.get_from_api(query).get('rate')

    def __str__(self):
        return f'{self.amount_sold} of {self.currency_sold} traded for {self.amount_bought
} of {self.currency_bought}'

class Period(models.Model):
    user_id = models.ForeignKey(User, on_delete=models.CASCADE)
    date_start = models.DateField(default=timezone.now)
    date_end = models.DateField()
    max_acts = models.PositiveSmallIntegerField()
    DAY = "D"
    WEEK = "W"
    MONTH = "M"
    window_choices = [(DAY,'Day'),(WEEK,'Week'),(MONTH,'Month')]
    acts_window = models.CharField(max_length=1,choices=window_choices,default=DAY)
    max_freq = models.PositiveSmallIntegerField()
    max_simultaneous = models.PositiveSmallIntegerField()
    use_shoulder = models.BooleanField(default=True)

    def get_absolute_url(self):
        return reverse("trader_journal:period_detail", kwargs={"pk": self.pk})

    def get_num_ops(self):
        return Operation.objects.filter(user_id=self.user_id,datetime__gte=self.date_start
,datetime__lte=self.date_end).count()

    def get_num_ops_open(self):
        return Operation.objects.filter(user_id=self.user_id,datetime__gte=self.date_start
,datetime__lte=self.date_end,is_open=True).count()

    def get_total_profit(self):
        profit = 0
        ops = Operation.objects.filter(user_id=self.user_id,datetime__gte=self.date_start,
datetime__lte=self.date_end).all()
        for op in ops:
            profit+=op.get_profit()
        return profit

    def limit_exceeded(self):
        ops = Operation.objects.filter(user_id=self.user_id,datetime__gte=self.date_start,
datetime__lte=self.date_end).all()
        for op in ops:
            if self.acts_window=='D' and ops.filter(datetime__day=op.datetime.day).count()
>self.max_freq:
```

```
                return True
            elif self.acts_window=='W' and ops.filter(datetime__week=op.datetime.week).cou
nt()>self.max_freq:
                return True
            elif self.acts_window=='M' and ops.filter(datetime__month=op.datetime.month).c
ount()>self.max_freq:
                return True
        if self.get_num_ops_open()>self.max_simultaneous:
            return True
        if self.date_end>=date.today() and self.date_start<=date.today():
            if Active.objects.filter(user_id=self.user_id).count()>self.max_acts:
                return True
        return False

    def __str__(self):
        return f'Started at: {str(self.date_start)}, ends (or ended) at: {str(self.date_en
d)}'
```

views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User
from django.contrib.auth.decorators import login_required
from django.utils.translation import ugettext_lazy as _
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy
from django.contrib import messages
from django.db import transaction
from django.views import generic
from django.db.models import Q
from django.http import Http404

from . import models, forms
import logging

logger = logging.getLogger(__name__)

@login_required
def index(request):
    """View function for home page of site."""
    user = request.user
    num_ops = 0
    cur_period = models.get_current_period(user)
    num_ops = user.operation_set.count()
    actives = user.active_set.order_by('-amount')[:5]
    total_worth = 0
    profit_made = 0
    initial_deposit = 0
    for act in actives:
        active_worth = act.get_in_usd()
        total_worth+= float(active_worth)
        if act.is_initial:
            initial_deposit+=float(active_worth)
```

```
        else:
            profit_made+=float(active_worth)
    context = {
        'num_ops': num_ops,
        'cur_period': cur_period,
        'actives':actives,
        'total_worth':total_worth,
        'profit_made':profit_made,
        'initial_deposit':initial_deposit,
    }
    return render(request, 'index.html', context=context)

@transaction.atomic
def register(request):
    if request.user.is_authenticated:
        messages.error(request,_("You're already registered!"))
        return redirect('trader_journal:index')
    if request.method == 'POST':
        user_form = forms.RegisterForm(request.POST)
        if user_form.is_valid():
            user = User.objects.create_user(user_form.data['username'], user_form.data['em
ail'], user_form.data['password'])
            user.save()
            messages.success(request, _('Your profile has been registered!'))
            return redirect('trader_journal:index')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        user_form = forms.RegisterForm()
    return render(request, 'registration/register.html', {
        'user_form': user_form
    })


@login_required
@transaction.atomic
def profile(request):
    if request.method == 'POST':
        user_form = forms.UserForm(request.POST, instance=request.user)
        if user_form.is_valid():
            user_form.save()
            messages.success(request, _('Your profile was successfully updated!'))
            return redirect('trader_journal:index')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        user_form = forms.UserForm(instance=request.user)
    return render(request, 'registration/profile.html', {
        'user_form': user_form
    })
```

```
@login_required
@transaction.atomic
def add_operation(request):
    if request.method == 'POST':
        op_form = forms.OperationForm(request.POST)
        if op_form.is_valid():
            operation = op_form.save(commit=False)
            operation.user_id = request.user
            period = operation.get_period()
            if period is not None:
                all_ops = models.Operation.objects.filter(datetime__gte=period.date_start,
 datetime__lte=period.date_end)
                if operation.is_open:
                    open_ops = all_ops & models.Operation.objects.filter(is_open=True)
                    if len(open_ops)>=period.max_simultaneous:
                        messages.warning(request, _('Open deal limit for period reached!')
)
                        period.limit_exceeded = True
                if period.acts_window=='D':
                    all_ops = all_ops & models.Operation.objects.filter(datetime__day=oper
ation.datetime.day)
                elif period.acts_window == 'W':
                    all_ops = all_ops & models.Operation.objects.filter(datetime__week=ope
ration.datetime.week)
                else:
                    all_ops = all_ops & models.Operation.objects.filter(datetime__month=op
eration.datetime.month)
                if len(all_ops)>=period.max_freq:
                    messages.warning(request, _('Operation limit for period reached!'))
                    period.limit_exceeded = True
                    period.save()
            operation.save()
            messages.success(request, _('Operation saved!'))
            return redirect('trader_journal:operations')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        op_form = forms.OperationForm()
    return render(request, 'trader_journal/operation_form.html', {
        'form': op_form
    })

class UserDetailView(generic.DetailView):
    def get_object(self, queryset=None):
        obj = super(UserDetailView,self).get_object()
        if not obj.user_id.id==self.request.user.id:
            raise Http404
        else:
            return obj

class UserUpdateView(generic.UpdateView):
```

```
    def get_object(self, queryset=None):
        obj = super(UserUpdateView,self).get_object()
        if not obj.user_id.id==self.request.user.id:
            raise Http404
        else:
            return obj

class UserDeleteView(generic.DeleteView):
    def get_object(self, queryset=None):
        obj = super(UserDeleteView,self).get_object()
        if not obj.user_id.id==self.request.user.id:
            raise Http404
        else:
            return obj

class OperationDetailView(UserDetailView):
    model = models.Operation

class OperationListView(generic.ListView):
    model = models.Operation

    def get_queryset(self):
        if self.request.user.is_anonymous:
            raise Http404
        else:
            return self.request.user.operation_set.all()

class OperationUpdate(UserUpdateView):
    form_class = forms.OperationForm
    model = models.Operation

class OperationDelete(UserDeleteView):
    model = models.Operation
    success_url = reverse_lazy('trader_journal:operations')

@login_required
@transaction.atomic
def add_active(request):
    if request.method == 'POST':
        act_form = forms.ActiveForm(request.POST)
        if act_form.is_valid():
            active = act_form.save(commit=False)
            active.user_id = request.user
            per = models.get_current_period(request.user)
            if per is not None:
                if per.max_acts<=request.user.active_set.count():
                    messages.warning(request, _('You have exceeded the current limit of ac
tives for the current period.'))
            if request.user.active_set.filter(currency=active.currency).exists():
                messages.error(request, _('You already have this active registered to your
 account. Please edit it instead of adding duplicates.'))
```

```
            else:
                active.save()
                messages.success(request, _('Active saved!'))
                return redirect('trader_journal:actives')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        act_form = forms.ActiveForm()
    return render(request, 'trader_journal/active_form.html', {
        'form': act_form
    })

class ActiveDetailView(UserDetailView):
    model = models.Active

class ActiveListView(generic.ListView):
    model = models.Active

    def get_queryset(self):
        if self.request.user.is_anonymous:
            raise Http404
        else:
            return self.request.user.active_set.all()

class ActiveUpdate(UserUpdateView):
    form_class = forms.ActiveForm
    model = models.Active

class ActiveDelete(UserDeleteView):
    model = models.Active
    success_url = reverse_lazy('trader_journal:actives')

def add_period(request):
    if request.method == 'POST':
        per_form = forms.PeriodForm(request.POST)
        if per_form.is_valid():
            period = per_form.save(commit=False)
            period.user_id = request.user
            if request.user.period_set.exclude(Q(date_end__lt=period.date_start)|Q(date_st
art__gt=period.date_end)).exists():
                messages.error(request, _('Overlapping periods are not allowed.'))
            else:
                period.save()
                messages.success(request, _('Period saved!'))
                return redirect('trader_journal:periods')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        per_form = forms.PeriodForm()
    return render(request, 'trader_journal/period_form.html', {
        'form': per_form
```

```
    })

class PeriodDetailView(UserDetailView):
    model = models.Period

class PeriodListView(generic.ListView):
    model = models.Period

    def get_queryset(self):
        if self.request.user.is_anonymous:
            raise Http404
        else:
            return self.request.user.period_set.all()

def update_period(request):
    if request.method == 'POST':
        per_form = forms.PeriodForm(request.POST)
        if per_form.is_valid():
            period = per_form.save(commit=False)
            period.user_id = request.user
            if request.user.period_set.exclude(Q(pk=period.pk)|Q(date_end__lt=period.date_
start)|Q(date_start__gt=period.date_end)).exists():
                messages.error(request, _('Overlapping periods are not allowed.'))
            else:
                period.save()
                messages.success(request, _('Period updated!'))
                return redirect('trader_journal:periods')
        else:
            messages.error(request, _('Please correct the error below.'))
    else:
        per_form = forms.PeriodForm()
    return render(request, 'trader_journal/period_form.html', {
        'form': per_form
    })

class PeriodUpdate(UserUpdateView):
    form_class = forms.PeriodForm
    model = models.Period

class PeriodDelete(UserDeleteView):
    model = models.Period
    success_url = reverse_lazy('trader_journal:periods')
```

urls.py

```
from django.urls import include, path
from . import views
from django.contrib.auth import views as auth_views

app_name = 'trader_journal'
urlpatterns = [
    path('', views.index, name='index'),
    path('register/',views.register, name='register'),
```

```
    path('profile/',views.profile, name='profile'),
    path('operations/',views.OperationListView.as_view(),name='operations'),
    path('operation/<int:pk>/', views.OperationDetailView.as_view(), name='operation_detai
l'),
    path('operation/create/', views.add_operation, name='operation_create'),
    path('operation/<int:pk>/update/', views.OperationUpdate.as_view(), name='operation_up
date'),
    path('operation/<int:pk>/delete/', views.OperationDelete.as_view(), name='operation_de
lete'),
    path('actives/',views.ActiveListView.as_view(),name='actives'),
    path('active/<int:pk>/', views.ActiveDetailView.as_view(), name='active_detail'),
    path('active/create/', views.add_active, name='active_create'),
    path('active/<int:pk>/update/', views.ActiveUpdate.as_view(), name='active_update'),
    path('active/<int:pk>/delete/', views.ActiveDelete.as_view(), name='active_delete'),
    path('periods/',views.PeriodListView.as_view(),name='periods'),
    path('period/<int:pk>/', views.PeriodDetailView.as_view(), name='period_detail'),
    path('period/create/', views.add_period, name='period_create'),
    path('period/<int:pk>/update/', views.PeriodUpdate.as_view(), name='period_update'),
    path('period/<int:pk>/delete/', views.PeriodDelete.as_view(), name='period_delete'),
    path('accounts/',include('django.contrib.auth.urls'))
]
```

Forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from django.utils import timezone
from django.utils.translation import gettext_lazy as _


from . import models, conv

class RegisterForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('username', 'email', 'password')

class UserForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('username', 'email')

class ActiveForm(forms.ModelForm):
    def clean_currency(self):
        data = self.cleaned_data['currency']
        if not conv.check_currency(data):
            raise ValidationError(_('This currency does not exist'))
        return data

    class Meta:
        model = models.Active
        fields = ('currency','amount','is_initial')
```

```
        labels = {'currency': _('Enter currency id'),'amount':_('Enter the amount of curre
ncy')}
        help_texts = {'currency': _('The currency id must be integreated into CoinAPI'),
        'amount':_('You can enter negative amounts of actives to signify a net loss.')}


class OperationForm(forms.ModelForm):
    def clean_datetime(self):
        data = self.cleaned_data['datetime']
        if data > timezone.now():
            raise ValidationError(_('Operations cannot occur in the future'))
        return data

    def clean_currency_bought(self):
        data = self.cleaned_data['currency_bought']
        if not conv.check_currency(data):
            raise ValidationError(_('This currency does not exist'))
        return data

    def clean_currency_sold(self):
        data = self.cleaned_data['currency_sold']
        if not conv.check_currency(data):
            raise ValidationError(_('This currency does not exist'))
        return data

    def amount_bought(self):
        data = self.cleaned_data['amount_bought']
        if data<=0 :
            raise ValidationError(_('Cannot buy a negative amount of a currency'))
        return data

    def amount_sold(self):
        data = self.cleaned_data['amount_sold']
        if data<=0 :
            raise ValidationError(_('Cannot sell a negative amount of a currency'))
        return data

    def clean_comission_percentage(self):
        data = self.cleaned_data['commission_percentage']
        if data<=0 or data>=100:
            raise ValidationError(_('Not a valid percentage'))
        return data

    class Meta:
        model = models.Operation
        fields = ('datetime','currency_bought','currency_sold','amount_bought','amount_sol
d','exchange_name','commission_percentage','is_maker','is_open')
        labels = {'currency_bought': _('Enter bought currency id'),'currency_sold': _('Ent
er spent currency id'),
        'amount_bought':_('Enter the amount of currency you received'),
        'amount_sold':_('Enter the amount of currency you spent')}
```

```
        help_texts = {'currency_bought': _('The currency id must be integreated into CoinA
PI'),
        'currency_sold': _('The currency id must be integreated into CoinAPI')}

class PeriodForm(forms.ModelForm):
    def clean_date_end(self):
        data = self.cleaned_data['date_end']
        if data<self.cleaned_data['date_start']:
            raise ValidationError(_('The period cannot end before starting'))
        return data

    def clean_max_acts(self):
        data = self.cleaned_data['max_acts']
        if data<=0:
            raise ValidationError(_('Not a valid amount of actives'))
        return data

    def clean_max_freq(self):
        data = self.cleaned_data['max_freq']
        if data<=0:
            raise ValidationError(_('Not a valid amount of total operations'))
        return data

    def clean_max_simultaneous(self):
        data = self.cleaned_data['max_simultaneous']
        if data<=0:
            raise ValidationError(_('Not a valid amount of open operations'))
        return data

    class Meta:
        model = models.Period
        fields = ('date_start','date_end','max_acts','acts_window','max_freq','max_simulta
neous','use_shoulder')
        labels = {'max_acts': _('Max actives'),
        'acts_window': _('Reset period'),
        'max_freq':_('Max in period'),
        'max_simultaneous':_('Max simultaneously open'),
        'use_shoulder':_('Use of shoulder')}
        help_texts = {'max_acts': _('How many actives will you own during the period'),
        'acts_window': _('How frequently does the total operation limit reset'),
        'max_freq':_('Limit on total operations in the period'),
        'max_simultaneous':_('Limit on open operations in the period'),
        'use_shoulder':_('Whether the use of shoulder is allowed')}
```

Conv.py

```
from django.core.exceptions import ObjectDoesNotExist
import requests

from .settings import API_KEY,API_URL
from datetime import date

'''Various convenience methods for working with the API'''
```

```
def get_from_api(query):
        url = API_URL+query
        headers = {'X-CoinAPI-Key' : API_KEY}
        response = requests.get(url, headers=headers)
        return response.json()

def get_exchange_names():
        return [exch.get('name') for exch in get_from_api('/exchange')]

def check_currency(currency_id):
        query = f'/exchangerate/{currency_id}/USD'
        result = get_from_api(query)
        if result.get('error') is not None:
                return False
        else:
                return True
```