Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК 004.75

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: “Програмна система забезпечення довіри в децентралізованих системах”
Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

МР.ІПЗ - 22.00.00.000

Студент

_____/Андрій БОЯРСЬКИЙ/_____
(підпис) (розшифровка підпису) (дата)

Науковий керівник

професор____/Віктор ШЕВЧЕНКО/____
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

_____Олексій БИЧКОВ_____
(підпис) (розшифровка підпису) (дата)

Київ 2022

Рішенням Екзаменаційної комісії №2

випускна кваліфікаційна робота студента

Андрія БОЯРСЬКОГО

захищена з оцінкою

_____

Голова Екзаменаційної комісії №2

професор, д.т.наук Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)
“___”_____ 20___р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
Андрію БОЯРСЬКОМУ
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи: “Програмна система забезпечення довіри в децентралізованих системах”
затверджена наказом вищого навчального закладу від „21” грудня 2021р. №8

2. Строк здачі студентом закінченої роботи 10 травня 2022р.
3. Вихідні дані до роботи
   - Дані про існуючі децентралізовані P2P системи та методи координації роботи вузлів
   - Різновиди атак на DLT системи, відомі і перспективні методи протидії.
   - Алгоритми глобального договору (консенсусу) в децентралізованих системах, способи досягнення високої операційної ефективності
   - Поточні інститути репутації та мотиваційні системи активних учасників DLT P2P мережі
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
   - Визначення мету, цілі, задачі та коло проблемних питань до вирішення.
   - Аналіз існуючих систем забезпечення довіри для DLT рішень, проведення їх класифікації, виявлення переваг та недоліків.
   - Визначення вимог до розроблюваної системи.
   - Проектування архітектури та окремих компонентів
   - Обгрунтування прийнятих технічних рішень
   - Розробка системи забезпечення довіри для децентралізованої піринговой мережі, демонстрація її можливостей та проведення тестування.
   - Визначення практичної цінності розробленого ПЗ та шляхів його покращення
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
   - Діаграма варіантів використання інтегрованої DLT системи
   - Діаграма послідовності обробки операції (транзакції)
   - Діаграма пов’язаних компонентів розроблюваної програмної системи
   - Діаграма організації класів та пакетів
   - Діаграма моделі реалізації шардингу
   - Діаграма фінальності операцій у DLT системі

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1 Аналіз існуючих рішень	Віктор ШЕВЧЕНКО	23.12.2021	24.01.2022
Розділ 2 Аналіз вимог і проєктування програмної системи	Віктор ШЕВЧЕНКО	25.01.2022	19.02.2022
Розділ 3 Програмна реалізація	Віктор ШЕВЧЕНКО	20.02.2022	25.04.2022

7. Дата видачі завдання 22 грудня 2022

Керівник _____ /Віктор ШЕВЧЕНКО/
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ /Андрій БОЯРСЬКИЙ/
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
Підбір і вивчення літератури по DLT системам	23.12.2021	виконано
Аналіз алгоритмів консенсусу децентралізованих систем	02.01.2022	виконано
Класифікація існуючих DLT систем	12.01.2022	виконано
Дослідження технічних особливостей EOS та Hyperledger Fabric	20.01.2022	виконано
Визначення проблемних питань, аналіз вимог та проектування системи забезпечення довіри	25.01.2022	виконано
Розробка механізму протидії атакам подвійної витрати, повторення операції, саботажу обробки операцій	08.02.2022	виконано
Програмна реалізація системи забезпечення довіри в децентралізованих системах	20.02.2022	виконано
Інтеграція розробленої системи в DLT P2P вузол та проведення випробувань	01.04.2022	виконано

Студент – магістр _____ /Андрій БОЯРСЬКИЙ/
(підпис) (розшифровка підпису)

Керівник роботи _____ /Віктор ШЕВЧЕНКО/
(підпис) (розшифровка підпису)

АНОТАЦІЯ *(на трьох мовах)*

Випускна кваліфікаційна магістерська робота: 82 с., 18 рис., 1 табл., 2 додат., 29 джерел.

Тема: програмна система забезпечення довіри в децентралізованих системах.

Об’єкт дослідження: P2P permissionless DLT мережі з високою продуктивністю, технології масштабування та стійкості DLT мереж.

Мета роботи: розробка та інтеграція системи забезпечення довіри для децентралізованих P2P мереж для забезпечення спільного координування операцій з інститутом репутації.

Предмет дослідження: алгоритми консенсусу в децентралізованих мережах, методи підтримки репутації активного вузла в мережі, типи атак на DLT вузли і методи протидії, шардинг та стиснення zk-snark.

Результати дослідження:

Визначено переваги та недоліки алгоритмів консенсусу POW, POS, dPOS в їх практичних реалізаціях. Розроблено і впроваджено систему забезпечення довіри в DLT P2P систему Dapos із захистом від саботажів і атак подвійної витрати і стимулюванням чесних учасників мережі.

Висновок

В ході дослідження було розроблено систему забезпечення довіри в програмних мережах без спільного центру координування, що дозволяє співпрацювати вузлам в недовіреній мережі з опором на репутацію і може бути інтегрована для координації виконання спільного завдання гетерогенними системами і мережами їх вузлів.

ТЕХНОЛОГІЯ РОЗПОДІЛЕНОГО РЕЄСТРУ, АЛГОРИТМ КОНСЕНСУСУ, БЛОКЧЕЙН, СИСТЕМ ГЛОБАЛЬНОГО ДОГОВОРУ, P2P МЕРЕЖА, DPOS, POS

АННОТАЦИЯ

Выпускная квалификационная магистерская работа: 82 с., 18 рис., 1 табл., 2 прил., 29 источников.

Тема: программная система обеспечения доверия в децентрализованных системах.

Объект исследования: P2P permissionless DLT сети с высокой производительностью, технологии масштабирования и устойчивости DLT сетей.

Цель работы: разработка и интеграция системы обеспечения доверия для децентрализованных сетей P2P для обеспечения совместного координирования операций с институтом репутации.

Предмет исследования: методы консенсуса в децентрализованных сетях, способы поддержания репутации активного узла в сети, типы атак на DLT узлы и способы противодействия, шардинг и сжатие zk-snark.

Результаты исследования:

Определены преимущества и недостатки алгоритмов консенсуса POW, POS, dPOS в их практических реализациях. Разработана и внедрена система обеспечения доверия в DLT P2P система Dapos с защитой от саботажей и атак двойного расхода и стимулированием честных участников сети.

Вывод

В ходе исследования была разработана система обеспечения доверия в программных сетях без общего центра координирования, позволяющая сотрудничать узлам в недоверенной сети полагаясь на репутацию и может быть интегрирована для координации выполнения общей задачи гетерогенными системами и сетями их узлов.

ТЕХНОЛОГИЯ РАСПРЕДЕЛЕННОГО РЕЕСТРА, АЛГОРИТМ КОНСЕНСУСА, БЛОКЧЕЙН, СИСТЕМА ГЛОБАЛЬНОГО ДОГОВОРА, P2P СЕТЬ, DPOS, POS

ANNOTATION

Final master's thesis: 82 pages, 18 figures, 1 table, 2 appendices, 29 sources.

Topic: software system to ensure trust in decentralized systems.

Object of research: P2P permissionless DLT networks with high performance, scaling technologies and stability of DLT networks.

Objective: Development and integration of a trust system for decentralized P2P networks to ensure joint coordination of operations with the institution of reputation.

Subject of research: consensus algorithms in decentralized networks, methods of maintaining the reputation of the active node in the network, types of attacks on DLT nodes and methods of counteraction, sharding and compression of zk-snark.

Results of the research:

The advantages and disadvantages of consensus algorithms POW, POS, dPOS in their practical implementations are determined. Developed and implemented a system of trust in the DLT P2P system Dapos with protection against sabotage and double-cost attacks and encourage honest network members.

Conclusion

The study developed a trust system in software networks without a common coordination center, which allows nodes in a nontrusted network to cooperate based on reputation and can be integrated to coordinate the joint task of heterogeneous systems and networks of their nodes.

DISTRIBUTED LEDGER TECHNOLOGY, CONSENSUS ALGORITHM, BLOCKCHAIN, GLOBAL AGREEMENT SYSTEM, P2P NETWORK, DPOS, POS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

BFT	-	byzantine fault tolerance
CA	-	certification authority
DAPP	-	decentralized application
DeFi	-	decentralized finance
DLT	-	distributed ledger technologies
dPOS	-	delegated proof of stake
ECC	-	elliptic curve cryptography
ICO	-	initial coin offering
POS	-	proof of stake
POW	-	proof of work
АОП	-	аспектно-орієнтоване програмування
БД	-	база даних
ІТ	-	інформаційні технології
ООП	-	об’єктно-орієнтоване програмування
ПЗ	-	програмне забезпечення

ВСТУП

Актуальність теми

Децентралізовані системи – системи що не мають центрального координувального вузла для виконання цілей системи, натомість використовують локальні дані та обчислення для досягнення спільної мети, можуть координуватися алгоритмами консенсусу (глобального договору з приводу коректності стану, даних) і дублювати, відновлювати дані, операції (повністю або частково), а також реалізовувати опосередковане централізоване керування в рамках децентралізованого голосування. Такі системи є стійкими до атак (DDoS, DoS, Man-in-the-Middle), краще захищають дані своїх користувачів, бо можуть їх, або не розповсюджувати, або шифрувати і обмежувати доступ, тому активно застосовуються для військових цілей, супутникового зв’язку, корпоративних комунікацій чи захисту інформації.

Підвид децентралізованих систем на базі DLT зазнав неабиякого розвитку починаючи з 2008 року, коли було вирішено задачу візантійських генералів для довільної кількості учасників (загального випадку) у рамках Біткоїна – P2P системи електронної готівки [1]. З тих пір класичні централізовані системи, як платіжні операції, кредитування, страхування стали активно децентралізуватись[2] на основі DLT, і актуальним є пошук можливостей по забезпеченню довіри в таких системах, бо доступ до них необмежений, і кожен учасник не довіряє за замовчуванням іншим, їм потрібен глобальний договір – консенсус.

Згідно звіту фокус групи International Telecommunication Union (ITU) від 1 серпня 2019 року [3], DLT системи можуть у перспективі використовуватись у різних секторах економіки: охорона здоров’я, фінанси, ланцюги поставок, державне управління, кіберспорт, розваги, проте перешкодами для цього є: недосконала нормативно-правова база, проблеми конфіденційності, довіри, масштабованості і продуктивності та відсутність стандартів.

Тому доцільним є розробка систем забезпечення довіри у DLT P2P мережах, оскільки DLT має значні переваги над централізованими аналогами, бо забезпечує

більш високу надійність, прозорість та незмінність всіх даних всередині мережі. Подібними рішеннями є Hyperledger та EOSIO.

Hyperledger Fabric [4] є приватним DLT рішенням з обов'язковою реєстрацією всіх учасників через Membership Service, що базується на центральному CA(Certification Authority), таким чином система стає закритою та авторизованою, що підходить для окремих бізнес задач, проте не для публічного використання, хоча продуктивність Hyperledger Fabric з реалізацією консенсусу PBFT в жовтні 2019 року досягає 14 тис. TPS [5]. Схожим на Hyperledger Fabric є реалізація R3 Corda, що хоч і підтримує шардинг, проте має низьку пропускну здатність ~600 TPS [6].

Альтернативним рішенням є EOSIO – операційна система на блокчейн протоколі з консенсусом DPOS та 4 тис. TPS [7], проте звіт дослідження ПЗ EOS компанією Whiteblock показав недостовірність цих заяв [8] (лише 250 TPS, високий рівень централізації та вразливість до атаки Сивілли і т.д.). Тим паче в травні 2020р. компанією Crypto Assets Opportunity Fund LLC було подано позов до Block.one про незаконне розповсюдження цінних паперів та обман інвесторів (протягом ICO) [9], тому використання такого ПЗ для бізнесу несе фінансові та організаційні ризики.

Згідно вище зазначеного, розробка системи забезпечення довіри для DLT в рамках даної магістерської роботи, що забезпечуватиме високопродуктивне досягнення консенсусу з публічною доступністю і прозорістю операцій, є цілком обгрунтованою.

Зв'язок роботи з науковими програмами, планами, темами

Відповідно до постанови Кабінету Міністрів України в вересні 2019р. створено Міністерство цифрової трансформації [10], однією із задач якого є легалізація і регуляція віртуальних активів [11] та розвиток Fintech і блокчейн технологічних напрямів [12]. Відповідний закон “Про віртуальні активи” уже прийнято Верховною Радою України [13].

Мета і задачі дослідження

Метою магістерської роботи є розробка системи довіри для розподіленої DLT P2P мережі вузлів, що забезпечить досягнення високопродуктивного консенсусу, захистить від недобросовісних чи зловмисних учасників мережі і реалізує

мотивацію для чесних учасників підтримувати роботу всієї системи в цілому і протидіяти порушенням чи втручанням в роботу системи.

Розроблювана система повинна реалізовувати сімейство алгоритмів консенсусу POS (доказ долі), забезпечувати 10 тис. TPS при 20 вузлах в мережі за наступних апаратних обмежень: 4 віртуальних ядра ~4Ghz, 4гб RAM, не мати системних обмежень та обов’язкових комісійних зборів як в EOS [14] та Pascal coin [15].

Для реалізації мети роботи ставляться наступні завдання:

- огляд існуючих високопродуктивних DLT систем
- дослідження моделей та алгоритмів консенсусу в децентралізованих мережах.
- визначення та формалізація підходів до протидії збоям і атакам в децентралізованих системах.
- аналіз вимог, проектування, реалізація, тестування, інтеграція розроблюваної системи забезпечення довіри в P2P систему.

Об’єкт дослідження: P2P permissionless DLT мережі з високою продуктивністю, технології масштабування та стійкості DLT мереж.

Предмет дослідження: алгоритми консенсусу в децентралізованих мережах, методи підтримки репутації активного вузла в мережі, типи атак на DLT вузли і методи протидії, шардинг та стиснення zk-snark.

Методи дослідження

Для дослідження існуючих алгоритмів консенсусу використовуються загальнонаукові методи аналізу, синтезу та моделювання. Для реалізації ПЗ в рамках даної роботи використовуються евристичні методи програмної інженерії, що включають методи об’єктно-орієнтованого проектування (UML) та аспектно-орієнтованого програмування. Для аналізу якості ПЗ використовуються емпіричні методи програмної інженерії, а саме статистичний аналіз метрик (статичний аналіз коду).

Наукова новизна отриманих результатів

Удосконалено механізм убезпечення від дублювання і повторного виконання операції в P2P системі (Double spending). Вперше реалізовано сумісність криптографічних еліптичних кривих secp256k1 та ed25519 для декількох типів аккаунтів в рамках DLT системи. Удосконалено систему заохочень і стягнень P2P вузлів (штрафи за пропуск чи ігнорування операцій, нагороди за підтримку роботи мережі).

Практичне значення одержаних результатів

Розроблену систему забезпечення довіри можна використовувати як самостійне рішення так і в складі існуючих децентралізованих P2P програмних комплексів для організації захищеної комунікації, координації з приводу спільного виконання операцій, заохочення чесних і накладання репутаційних та фінансових стягнень з недобросовісних чи зловмисних учасників мережі, протидії атакам дублювання операцій, підробки автентичності, видавання зловмисним вузлом себе за іншу особу, умисним чи випадковим збоям по валідації кінцевого стану спільних операцій.

Публікації

Проблематика та результати роботи опубліковані:

1. У збірнику тез VII Всеукраїнської науково-практичної конференції молодих науковців «Інформаційні Технології – 2020»

2. У збірнику тез 6-ої Східно-Европейської конференції “Математичні та програмні технології Internet of Everything”

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Основні задачі

Метою даного розділу є:

• огляд літератури, наукових статей, доповідей, тез конференцій, дисертацій та інших наукових робіт з напрямку Fintech, DLT, blockchain, алгоритми консенсусу POW, POS, DPOS, способи та корисні моделі покращення продуктивності DLT систем;

• аналіз існуючих DLT систем з приватним авторизованим доступом для організацій на базі CA (permissioned) і відкритим загальним доступом без необхідності ідентифікації (permisionless); визначення переваг і недоліків;

• визначення кола невирішених питань в побудові DLT систем для забезпечення довіри в децентралізованих системах та постановка задачі на дослідження і розробку.

1.2. Огляд літератури

Характеристики та типові компроміси, що приймаються у DLT системах (продуктивність і безпека, стійкість до вразливостей і аудиторність, величина блоку і консистентність), приведено у роботі [16]. Продуктивність п’яти популярних DLT (Tendermint, Ripple, Corda 2 версій та Hyperledger Fabric) для потреб IoT (інтернет речей) досліджувалась в рамках статті [17].

Основні ідеї та практичні варіанти використання біткоїна, а також опис схожих, конкуруючих криптовалют приводиться у книзі А. Антоноуполоса Mastering Bitcoin: Unlocking Digital Crypto Currencies [18].

У книзі Mastering Blockchain: Distributed ledger technology, decentralization and smart contracts explained [19] автором описується зріз всієї DLT екосистеми на 2018 рік з описом технічних особливостей технології блокчейн, смарт-контрактів,

токенізованих, нетокенізованих мереж, сайдчейнів, розподілених реєстрів, консенсусних механізмів та децентралізованих додатків на протоколах першого, другого і третього рівня. Особлива увага приділяється криптографічному захисту: асиметричне шифрування та генерація приватних і публічних ключів (RSA, ECC), коди аутентифікації повідомлення та їх застосування, функції хешування і цифрового підпису і їх роль в захисті від подвійного витрати, повторення транзакцій (replay protection) та генерації адреси в мережі.

Фахад Салех з університету Макгілла у своїй роботі “Blockchain Without Waste: Proof-of-Stake” [20] описує першу формальну економічну модель консенсусу POS у порівнянні з POW при цьому приводить параметри мережі, при яких вузли будуть найшвидше досягати консенсусу (необхідність обмеження кількості вузлів та мінімальної частки стейку для форджингу)

Повністю математично верифікована реалізація консенсусного протоколу POS (Ouroboros) для криптовалюти Cardano описана в доповіді на конференцію “Advances in Cryptology – Crypto 2017” [21]. В роботі основна увага присвячується гарантії безпеки алгоритму з протидією таким атакам, властивим POS-подібним консенсусам як: Pre computing attack (властивість ранніх реалізацій POS щодо визначення часу та вузла, що згенерує наступний блок заздалегідь з детерміністичної моделі часу), Nothing at stake attack (атака одного відсотку стейка, що супроводжується генеруванням блоків на декількох форках з вибором найбільш підходящого для зловмисника в певний проміжок часу), Spent stake attack (полягає у витраті стейку одним аккаунтом і створенням форку, до моменту витрати стейку — подвійна витрата), Long/Short range attack (при компрометації ключів форджерів у зловмисника є можливість почати свій форк від генезисного блоку чи блоку, де стейк у скомпрометованих ключів був найбільшим і таким чином змусити всю мережу підтримувати “змінений” блокчейн).

Плюси та мінуси POW, POS і частково DPOS подані у науковій статті О. Ващука і Р. Шувара [22], де в порівняльній таблиці наводяться типи атак на алгоритми консенсусу та вразливість для POS, POW і DPOS в їх еталонних реалізаціях без додаткових схем захисту та пом'якшувальних заходів. У статті

допущені деякі неточності з приводу Long range attack на POW консенсус та не розглядаються алгоритми “переключення” блокчейна на інший форк (rollback): (Heaviest chain, eventual finality), (PBFT, instant finality), алгоритм POW визначається найбільш стійким до атак, що не зовсім вірно, оскільки стійким він буде лише у випадку використання специфічної хеш-функції, відмінної від інших POW мереж, щоб майнери не могли застосувати атаку 51% з більш потужної мережі.

В той час як в статті “A survey on Long-Range Attacks for Proof of Stake Protocols” [23] авторами всебічно розглядаються принципи фінальності транзакцій, можливі шляхи атаки на POS консенсус та розробляються методи протидії більшості атак, такі як: Правило найдовшого ланцюга (Longest/Heaviest chain), чекпоїнт (точка фінальності блокчейну), що зміщується в часі; еволюція криптографічних ключів (заборона підписувати приватним ключем старі повідомлення через введення спеціальних KDF (key derivation function), що прив'язуються до епохи); транзакції прив'язані до попереднього блока (забороняє дублювати транзакції на форках); Правило повноти (ланцюг у якого найбільша щільність блоків і стала швидкість їх генерації є правильним); економічна фінальність (штрафування валідаторів за подвійний підпис блоків на однаковій висоті — захист від форків).

Токенізація активів з використанням блокчейну наводиться у працях [24-25], де описуються підходи до токенізації (модель OAP — open asset protocol) та реалізація у вигляді PBT (Policy-Backed Token), що застосовується в сферах страхування, а також наводяться різні види токенізованих активів, такі як: equity token (актив, що являється представленням акцій компанії в блокчейні і надає право власнику мати право голосу в компанії, отримувати дивіденди і визначати стратегію розвитку); security token – надає право на частину прибутку компанії, належність активу до securities визначають по Howey тесту, де задають 4 питання, на кожне з яких повинна бути відповідь “так”: ти інвестуєш гроші?, ти очікуєш прибуток?, ти інвестуєш в звичайне підприємство?, чи залежить прибуток від третіх сторін?; utility token – дають доступ до користування якимось сервісом чи платформою, або надають певну перевагу у користуванні і оскільки їх кількість обмежена, то в залежності від популярності сервісу чи компанії їх ціна може змінюватись, це

єдиний вид активів що отримується в процесі ICO і не підпадає під регулювання законами про цінні папери і не надає інвесторам ніяких гарантій та не забезпечує захищеність прав своїх утримувачів.

Моделі транзакцій з нульовими комісіями були реалізовані в криптовалютах EOS (стейкінг балансу більше 10 EOS) [14], NANO [26] (POW підбір значення nonce для кожної транзакції алгоритмом HashCash для захисту від спаму), IOTA [27] (POW підтвердження двох транзакцій в дереві перед відправкою однієї власної транзакції), Pascal Coin [15] (на кожен п’ятихвилинний блок у кожного аккаунта є одна безкоштовна транзакція). Всі ці способи мають недоліки, оскільки у випадку IOTA для інтернету речей виконувати POW для підтвердження транзакцій є дуже неефективним, оскільки IoT девайси не є високопродуктивними (мають 8-16 бітні контролери на частоті до 100Мгц і мало пам'яті до 32 Мб) а також дуже часто є автономними, тому POW істотно впливає на розряд батареї; Pascal Coin має недолік у вигляді eventual finality та тривалого часу підтвердження транзакцій (як біткоїн), при цьому група аккаунтів може закривати блоки лише спам транзакціями без комісії, що робить цю мережу вразливою до DDOS атак. NANO також використовує POW консенсус, що не є високоефективним, а для користувача, що відправляє багато транзакцій – неможливо досягти навіть 50 TPS, оскільки кожна транзакція потребує як мінімум 3-4 секунд генерування nonce потужним центральним процесором рівня Intel i7 4770k та краще, таким чином сервіс-провайдери повинні купувати чи орендувати потужне апаратне забезпечення і делегувати POW роботу на нього, що потребує додатково спеціального програмного забезпечення, тому є економічно невигідним. EOS взагалі не має звичайного механізму сплати комісій, тому Xodus wallet реалізували його окремо [14], транзакції ж без комісії потребують стейкінгу (заморозки) 10 EOS, що не дозволяє використовувати даний механізм для мікроплатежів, також такий спосіб не захищає від спам-транзакцій.

В whitepaper BitShares [28] вперше на високому рівні Д. Ларімером був описаний алгоритм консенсусу DPOS та його основні відмінності від POW, при цьому зазначено основні оптимізації, що виконувались для досягнення продуктивності в 3 тис. TPS: паралельні криптографічні функції, розподіл валідації

на залежну від стану та незалежну, більшість core даних в оперативній пам’яті, основна бізнес логіка виконується в новому потоці.

Варто зазначити, що у проаналізованих роботах по DPOS консенсусам наводяться розмиті моделі підтримки репутації вузла в мережі, методи протидії атакам подвійної витрати, повторення операції, підробки автентичності і т.д, що ускладнює реалізацію такого типу систем. Не приділялось уваги дослідників до проблематики стандартизації протоколів роботи DLT систем і можливості їх інтеграції в існуючі децентралізовані мережі. Також мало уваги приділено бізнес-потребам компаній і сервісів, що використовують DLT для розрахунків та обліку (невідворотність транзакцій та висока пропускна спроможність). Розрізнена інформація подається про класифікацію DLT проектів за технічними специфікаціями, як правило в рамках наукових робіт досліджувалась лише економічна складова.

Тому доцільним є складання порівняльної класифікація DLT систем і розробка моделі забезпечення довіри в децентралізованих системах як стандартизованої, виділеної частини ПЗ DLT системи.

1.3. Класифікація DLT систем

На час написання даної роботи існує понад 19500 публічних DLT проектів представлених криптовалютами [2], кожен з яких націлений на вирішення окремої задачі чи проблеми, багато з цих проектів дублюють ідеї один одного і намагаються презентувати одну й ту саму технології з різних сторін для отримання короткотривалого конкурентної переваги. Більшість криптовалют є клонами (форками) існуючих раніше проектів таких як Bitcoin, Ethereum (найбільше) з мінімальним переліком змін.

DLT проекти можна класифікувати за видом консенсусу:

1. POS (доказ долі) – NXT, Cardano, Peercoin.
2. POW (доказ роботи) – Bitcoin, Litecoin, Ethereum.
3. DPOS (делегований доказ долі) – Lisk, Steem, Minter, Bitshares.

4. Гібридний (декілька консенсусів одночасно) – EOS (aBFT + DPOS).

За часом настання консенсусу:

1. Синхронний (BFT-подібні протоколи) – досягається лише у випадку погодження більшості вузлів (66% + 1).

2. Асинхронний (POW/POS/DPOS протоколи) – досягається у індивідуальному порядку кожним вузлом окремо і синхронізується з іншими вузлами мережі, породжуючи форки, якщо інші вузли досягли інакшого консенсусного рішення в той же самий час.

За рівнем функціонування обігових засобів:

1. Першого рівня (нативні) – базові розрахункові засоби для DLT мереж (для Bitcoin — BTC, для Ethereum – ETH), функціонують на власному блокчейні.

2. Другого рівня (кольорові монети, токени) – випущені на сторонньому блокчейні першого рівня, не мають власного консенсусного механізму, мережі та реєстру даних, а використовують інфраструктуру базової мережі, де вони випущені (ERC20 токени Ethereum – USDT, Paxos Standard, Maker)

За призначенням:

1. Трейдинг та інвестиційна діяльність (FTX Token, Ravencoin)

2. Платежі та розрахунки (Bitcoin, Digibyte, Monero)

3. Страхування і кредитування (Nexo, Crypto.com)

4. Соціальні мережі (Steemit, Status, Hive)

5. Охорона здоров’я (Solve, Stem cell coin, MediBloc)

6. Розваги (Theta, Chiliz, Bora)

7. Децентралізовані сховища (Siacoin, RSK, Storj)

8. Краудфандинг (IHF, Wings, Tokenomy)

9. Загального призначення (NXT, Ethereum, Algorand)

За типом забезпечення:

1. Незабезпечені – не мають ніяких резервів чи еквівалентах замінників, що можуть бути отримані замість існуючої одиниці обігу.

2. Крипто-забезпечені – мають забезпечення у вигляді інших криптовалют або їх корзини (збалансований деривативний індекс).

3. Фіато-забезпечені – забезпечені фіатними валютами, їх похідними, реальними товарами (дорогоцінні метали, нафта), акціями, опціонами, облігаціями, свопами і т.д.

За мірою вираження цінності:

1. Стейблкоїн – цінність розрахункової одиниці в DLT мережі дорівнює цінності реального активу, як правило валюти (долар, євро) – Tether USD, Paxos Standard, Gemini USD, Binance USD.

2. Монета плаваючої цінності – цінність залежить від балансу попиту і пропозиції та ліквідності (EOS, Ethereum, ZEC та ін.).

За ступенем анонімності:

1. Публічно-прозорі (Bitcoin, Ethereum) – транзакції, адреси акаунтів та їх баланс доступні для перегляду і аналізу будь-якому користувачу мережі.

2. Анонімні – Zero cash, Monero – використовуються криптографічні методи маскування відправника та отримувача транзакцій а також їх балансів за допомогою кругових підписів, zero knowledge proofs і т.д.

3. Частково анонімні (реалізують лише частину концепцій анонімності) – Dash, Nxt – як правило використовуються протоколи другого рівня, такі як Coin mixing та Coin Join.

За типом використовуваного реєстру даних:

1. Блокчейн (Bitcoin, Zcash, Ethereum) – реєстр блоків з транзакціями кожен з яких пов'язується з попереднім на базі хеша, формуючи повністю верифікований ланцюг даних; кожен новий вузол завантажує блокчейн з генезисного блоку верифікуючи кожен блок один за одним.

2. DAG (IOTA, Byteball) – замість згрупованих блоків транзакцій, самі транзакції використовуються як блоки даних і затверджуються мережею на базі зв’язування з іншими транзакціями утворюючи ациклічний направлений граф – DAG, кожна транзакція отримує підтвердження у вигляді POW роботи.

3. Non-blockchain (Cosmos Hub) – відмінність від звичайного блокчейн рішення полягає у тому, що для досягнення консенсусу новим вузлам не потрібно повністю повторювати верифікацію і завантаження всіх блоків і транзакцій.

За моделлю фінальності транзакцій:

1. Миттєва фінальність (instant finality) – транзакція вважається невідворотною (irreversible), якщо вона потрапляє в блок, що підтверджений мережею, такий механізм характерний для сімейства реалізацій BFT протоколів.

2. Ймовірна фінальність (eventual finality) – транзакція вважається такою, що має високу ймовірність бути невідворотною, якщо для блоку, в якому вона була прийнята, отримано деяка кількість підтверджень (були поверх додані блоки, що базуються на даному блоці), таку схему подано на рис.1.1.

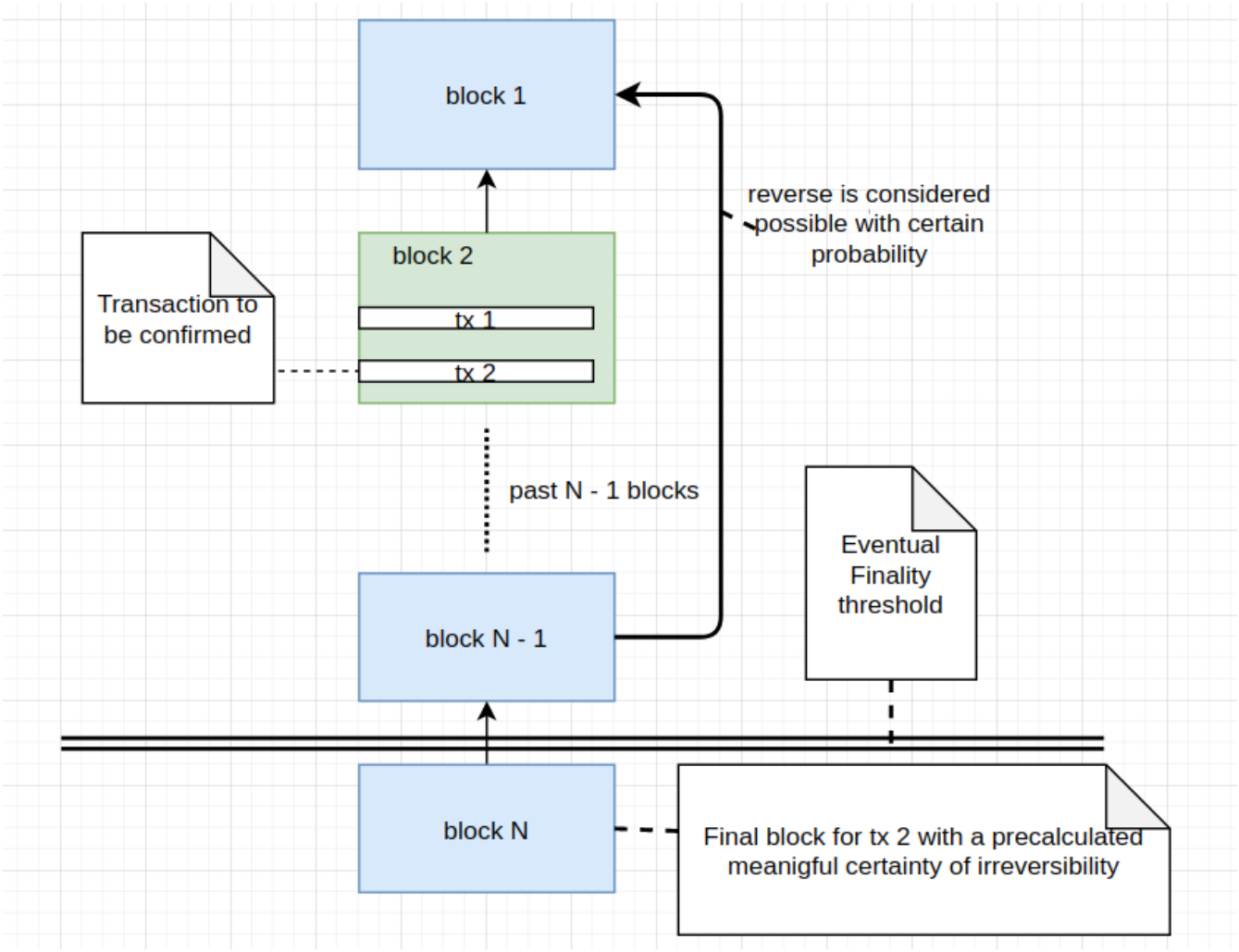


Рис. 1.1. Схема eventual finality

Eventual finality характерна для більшості DLT проектів (Bitcoin, Ethereum, ін.).

За способом генерації блоків:

1. Майнинг – механізм вирішення криптографічної головоломки для знаходження такої відповіді (як правило числа nonce) за обмежений проміжок часу,

що задовольнить певному складно досяжному правилу – для біткоїна це подвійне хешування алгоритмом SHA-256 числа та контенту блока, що повинно відповідати поточній складності генерування блоку (кількість нулів на початку хеша).

2. Стейкінг – це процес замороження балансу аккаунта, що дає можливість прийняти участь у процесі генерації і валідації блоків пропорційно до загального балансу затраченого мережею для стейкінгу. Якщо аккаунт вносить до стейкінгу 20% від загального об’єму стейкінгу (total staking power), то він в загальному випадку буде генерувати 20% всіх блоків і отримувати за них винагороду.

3. Голосування – учасники мережі поділяються на генераторів блоків та виборців, виборці можуть обирати у вільному форматі, за кого з генераторів блоків віддати свій голос (голосами може вважатися внутрішня валюта чи актив), в залежності від кількості голосів у кожного з генераторів блоків є пропорційний шанс згенерувати блок і отримати винагороду, яка потім може розділятися між виборцями.

4. Компонування – механізм характерний для Ardor, де кожний дочірній блокчейн для підтвердження транзакцій транслює їх у вигляді зкомпонованих транзакцій або консенсусних доказів чи повідомлень у батьківський блокчейн. За трансляцію може відповідати будь-який користувач, що зацікавлений у отриманні нагороди в валюті дочірньої мережі в обмін на валюту батьківської мережі (Ardor – Ignis).

За необхідністю аутентифікації:

1. Permissionless – не потребують аутентифікації та узгодження з контролюючою організацією додавання нового учасника, система відкрита для будь-кого, до такої категорії належить більшість DLT проектів загального призначення (Bitcoin, Ethereum, Neo, Omni та ін.).

2. Permissioned – процедури додавання, зміни і виходу учасників регламентується спеціальними “політиками використання”, що задаються експлуатуючою організацією при запуску DLT рішення. До таких систем належить сімейство продуктів Hyperledger: Fabric, Corda, Sawtooth.

За можливостями масштабування:

1. Сайдчейни – це залежні DLT мережі, що комунікують з батьківським блокчейном напряму без необхідності зміни батьківського блокчейну – це мережі другого рівня які працюють поверх батьківського.

2. Дочірні мережі – частково залежні системи що використовують батьківську як єдину точку синхронізації, відновлення і розслідування (пошук вузлів, що поводились зловмисно), проте працюють окремо підтримуючи власний блокчейн.

3. Шардинг – розбиття блокчейну на невеликі складові частини (шарди) між робочими вузлами для пропорційного збільшення пропускної здатності за рахунок зменшення кількості блоків і транзакцій, що необхідні оброблюватись кожним вузлом – часткова верифікована реплікація. Ця технологія є експериментальної і ще досі не була реалізована практично, оскільки породжує багато вразливостей. Ethereum Casper + Beacon chain, що відповідають за Ethereum 2.0 та Nighshade протоколи, частково реалізують шардинг, проте вони не є production-ready. Теоретична схема шардингу наведена на рис.1.2.

За методами сплати комісійних зборів:

1. По розміру транзакції – транзакція представляється у вигляді послідовності байтів і чим більше байтів займає транзакція, тим більше коштує її відправка. (Bitcoin, Digibyte).

2. По типу операції – найпростіший в реалізації, в залежності від типу операції, до якої відноситься транзакція, вона оцінюється сталою величиною комісії (NXT).

3. По ресурсним затратам – в залежності від кількості необхідних ресурсів для виконання транзакції: такти процесора, пам’ять, кількість швидких та повільних команд (Ethereum, EOS).

4. За навантаженістю мережі – в залежності від поточного стану DLT мережі (кількість транзакцій в пулі, частота прийняття нового блоку, сила генераторів блоків) збільшується або зменшується фінальна комісія на базі поправочного коефіцієнту за формулою (1.1). Правила налаштувань поправочних коефіцієнтів закладаються в механізмі консенсусу.

5. Система зобов'язань – відправник транзакції зобов'язується виконати якусь роботу (IOTA) чи внести вклад (EOS) перед безпосереднім відправленням транзакції в мережу, при цьому комісія з відправника не стягується, якщо він виконав зобов'язання коректно і його доказ пройшов перевірку у вузлів-валідаторів.

6. Комбіновані системи – можуть включати декілька видів сплати комісій залежно від типу транзакції, відправника чи зовнішніх обставин (Ethereum фіксовані по типу операції платежі та по ресурсам – функції смарт-контракту).

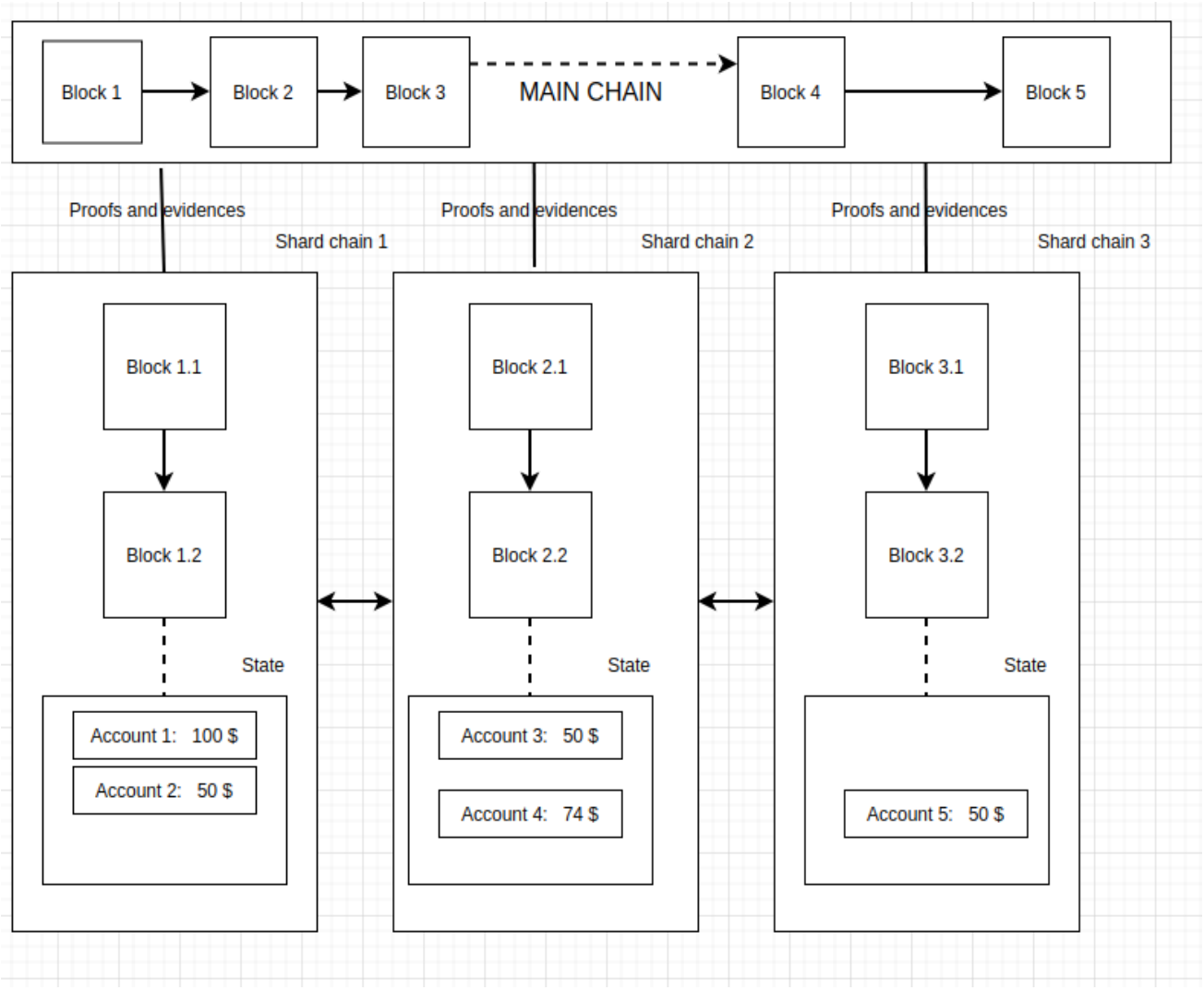


Рис. 1.2. Загальна схема блокчейн-шардінгу

$$F_t = max(0, F_b + F_p \times K), K \in [-10,10] \quad (1.1)$$

де F_t – фінальний розмір комісії,

F_b – базовий розмір комісії при нормальних умовах,

F_p – допустимий розмір комісії, який можна збільшувати чи зменшувати,

K – поправочний коефіцієнт.

Отже у ході проведення класифікації визначено основні типи DLT систем та принципи їх функціонування, для подальшого вирішення задачі магістерської роботи необхідно провести end-to-end аналіз популярних високопродуктивних реалізацій DLT та визначити їх плюси і мінуси в забезпеченні довіри між вузлами децентралізованої мережі.

1.4. Дослідження існуючих реалізацій DLT

Центр розвитку інформації та промисловості (CCID), при Міністерстві промисловості та інформаційних технологій Китаю, опублікував рейтинг криптовалют в середині квітня 2020-го р. згідно з якого перше місце посіла платформа EOS (156.1 балів), друге із значним відставанням Tron (138.4), третє відповідно Ethereum (136.4). Оскільки EOS є найуспішнішим ICO в світі (було зібрано понад 2 млрд. доларів, для порівняння Телеграм у ході 2 раундів приватного ICO зібрав 1.7 млрд доларів), а також реалізовує передові технічні концепції в DLT системах (гібридний aBFT + DPOS консенсус, розподілені обчислення, смарт-контракти, безкоштовні транзакції), тому варто розглянути EOS більш детально на предмет продуктивності, безпеки і застосовності для мікроплатежів.

EOS є представником публічних DLT систем загального призначення, орієнтованих на масового користувача, в той час як Hyperldedger [4] орієнтований на приватне використання з permissioned моделлю доступу використовуючи PKI (Public key infrastructure) та CA. Hyperledger розвивається Linux Foundation в рамках декількох реалізацій: Fabric, Iroha, Corda, Sawtooth. Hyperledger Fabric є production-ready DLT рішенням, що може використовуватись для потреб бізнесу, де необхідна приватність і конфіденційність. Оскільки Hyperledger уже впроваджувався для реалізації ланцюгів поставок (supply chains) та платформи фінансової торгівлі, в тому числі у Wallmart [29]– найбільшої компанії по доходам у світі, то доцільно

розглянути можливість його застосування для потреб DeFi з підтримкою мікротранзакцій і порівняти його з EOS.

В висновку по Hyperledger Fabric можна зазначити, що цей фреймворк для створення DLT систем надає широкі можливості по створенню приватних блокчейнів (модульність, конфіденційність, обмеженість доступу, відсутність форків та фінальність всіх транзакцій), проте вводить додаткові абстракції і функціональні складності, що сильно заважають розгортанню компактних мереж на Hyperledger, оскільки необхідно як мінімум 3 види пірів (committing peer, endorsement peer, ordering peer) та membership service і CA для нього, також в силу приватної природи Hyperledger Fabric ніяк не захищений від спам-транзакцій, якщо не вводити примусову ідентифікацію. Також Fabric є уразливим до затримки генерації і розповсюдження блоків і не може забезпечити необхідний рівень консистентності для розгортання в критичних середовищах. Тому використання Hyperledger Fabric для мікроплатежів не є ефективним і слід розглянути інші DLT реалізації.

Особливістю EOS є високий ступінь централізації (всього 21 валідатор) та механізми цензури, (привілейовані смарт-контракти) що значно зменшують привабливість цієї платформи для розгортання додатків DeFi, оскільки порушують принципи DLT систем. Також варто зазначити, що EOS підтримує базу даних форків, оскільки для часу блока в 500 мс може з’являтися багато нефіналізованих ланцюжків, що починає порушувати принцип алгоритмічної кінечності транзакцій і значно підвищує час на їх підтвердження, при цьому необхідність перемикання між форками приводить до втрати транзакцій і зменшення продуктивності, тому атаки з слабкими валідаторами можуть стати потенційною проблемою стабільного функціонування мережі.

Для наглядного порівняння характеристик EOS та Hyperledger було розроблено порівняльну табл.1.1. Таблиця розподіляє властивості децентралізованих продуктів за розробленою в підрозділі 1.3 класифікацією, вводячи додаткові характеристики по підтримуваним консенсусам, продуктивності (кількість транзакцій в секунду) та недолікам і особливостям кожної з систем.

Таблиця 1.1

Порівняльна характеристика EOS та Hyperledger Fabric

Характеристики	EOS	Hyperledger
Підтримувані консенсуси	aBFT + DPOS, реалізація додаткових – не передбачена	PBFT, RBFT, Plenum, інші необхідно реалізовувати окремо
Час блоку	0.5 мс	> 5 с
Фінальність транзакцій	Миттєва	Миттєва
Підтримка форків	+	-
Максимальний розмір блоку	1 мб (конфігурується)	без обмежень, конфігурований
Тип дистрибуції	Готова платформа	Фреймворк
Продуктивність (пікова)	4000 TPS	3000 TPS
Комісійні збори	Зобов'язання у вигляді стейкінгу	Залежить від реалізації, може бути будь-яким
Доступність	Публічний	Приватний
Сховище даних	Блокчейн, бд форків	Блокчейн, state db, ordering реєстр
Генерація блоків	Голосування + детерміновані графіки	Голосування
Ступінь анонімності	Відкритий	Частково анонімний (приватні канали)
Обіговий засіб	токени EOS першого рівня	активи другого рівня
Призначення	Загального призначення	Загального призначення
Можливі вразливості	Часті форки, змова валідаторів, цензура через привілейовані контракти	Спам невалідними транзакціями
Недоліки	Високий рівень централізації, необхідність стейкінгу для мікроплатежів	Складність запуску компактних мереж (високі початкові затрати)

З приведеної таблиці видно, що Hyperledger Fabric і EOS схожі в більшості пунктів,оскільки забезпечують достатньо високий рівень продуктивності та швидку фінальність транзакцій реалізовуючи при цьому BFT протокол консенсусу, що є доведеним та повністю верифікованим способом взаємодії в DLT системах. Проте принципова різниця з’являється у недоліках та вразливостях, наприклад схильність до форків у EOS і його централізація слабо підходять для токенізації активів для цифрової економіки, а Hyperledger вимагає складної конфігурації системи взаємодіючих пірів, що фактично за рахунок складності не покращують продуктивність (як видно з порівняння з EOS), тому доцільним є розробка окремої системи забезпечення довіри в децентралізованих системах, що зможе інтегруватись в існуючі DLT системи, вбере в себе переваги EOS і Hyperledger і зможе реалізувати інститут репутації і мотивації учасників.

1.5. Коло проблемних питань до вирішення

У ході аналізу здійсненого у попередніх підрозділах можна виокремити наступний список невирішених питань:

1. Універсальна система забезпечення довіри — в Hyperledger та EOS є спільна проблема неможливості безпосередньої інтеграції в існуючі децентралізовані системи, вони замкнуті самі на собі і модифікуються лише організаціями - учасниками процесу розробки.

2. Сумісність криптографічних систем – Hyperledger підтримує криптографічні модулі, проте вони є унікальними для кожного з каналів і не можуть працювати одночасно, EOS прив’язаний до однієї криптографічної системи.

3. Вразливість до форків та деградація продуктивності – властиві EOS через брак часу синхронізації.

1.6. Висновки до розділу

У ході проведеного дослідження було здійснено аналіз літератури, наукових статей за темами продуктивності існуючих DLT рішень приватного та публічного рівнів доступу; принципи та системи забезпечення довіри та їх недоліки; способи атаки та POS, POW, DPOS консенсуси та методи протидії і пом’якшення впливу цих атак на роботу мережі.

Було розроблено класифікацію DLT проектів за такими параметрами як: фінальність транзакцій, анонімність, тип консенсусу, сфера використання, метод генерації блоків , підтримка варіантів масштабування, методи сплати комісій і т.д.

Досліджено сучасні DLT системи EOS та Hyperledger Fabric, викладено принципи досягнення консенсусу і взаємодії вузлів у цих мережах, а також наведено недоліки цих програмних продуктів та складено відповідну порівняльну таблицю, що дозволило виявити високий ступінь подібності цих проектів у надаваних користувачу можливостях, також визначено можливі вразливості у цих системах і доведено їх недостатню придатність для реалізації універсальної системи забезпечення довіри для існуючих децентралізованих систем.

У ході аналізу було визначено коло проблем, що їх необхідно вирішити у подальшій розробці даної магістерської роботи.

РОЗДІЛ 2
АНАЛІЗ ВИМОГ І ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1. Специфікація вимог

Щоб формально визначити вимоги до системи, що необхідно реалізувати, було розроблено специфікацію вимог, що відповідає стандарту ISO/IEC/IEEE 29148:2018 з деякими спрощенням секцій “Вступ”, “Верифікація”, “Додаткова інформація”, оскільки метою даної магістерської роботи є лише розробка самого ПЗ без його формальної верифікації (верифікувати прототип немає сенсу з точки зору часових та людських ресурсів).

2.1.1. Вступ

Дана специфікація призначена для розробки вимог до програмного забезпечення системи забезпечення довіри в децентралізованих системах з довільною кількістю учасників. Розробка вимог провадиться до всього комплексу ПЗ DLT вузла, в який інтегрується система забезпечення довіри для можливості наглядної демонстрації її можливостей. Оформлення специфікації провадиться у відповідності до стандарту ISO/IEC/IEEE 29148:2018.

2.1.1.1 Ціль

Основною задачею цієї специфікації є високорівневий опис функціональних та нефункціональних вимог, що дозволить розробникам ПЗ системи звертатися до пунктів цієї специфікації у разі виникнення протиріч чи спірних питань з приводу реалізації чи інтеграції функцій системи; дозволить використовувати вимоги в цієї специфікації для їх ідентифікації, реалізації та тестування, а також відслідковування прогресу реалізації продукту в цілому; буде надавати загальне представлення про продукт для нових учасників, що пришвидшить їх адаптацію та дозволить їм швидше приступити до виконання задач.

2.1.1.2 Межі проекту

Система дозволяє користувачам:

1. Здійснювати P2P трансфери нативними токенами.

2. Делегувати оплату зборів за транзакції платникам послуг.

3. Відправляти зашифровані P2P повідомлення.

4. Проводити моніторинг стану P2P мережі.

5. Перевіряти реєстр виконаних транзакцій з балансом аккаунту.

6. Створювати аккаунти, сумісні з bitcoin та ethereum; для вузла-валідатора; нативний аккаунт мережі.

7. Голосувати за декілька вузлів-валідаторів та відкликати голоси.

8. Запускати власний вузол-валідатор та контролювати його роботу.

ПЗ системи складається з трьох компонентів:

1. TS – система забезпечення довіри для децентралізованої мережі, що захищає від атак подвійної витрати, повторення операцій, підробки автентичності вузла і забезпечує виконання стягнень і мотивації учасників.

1. DLT вузол – ПЗ, в яке інтегрується TS, що реалізує P2P рівень взаємодії з такими ж вузлами, забезпечує підтримку і виконання консенсусу децентралізованої мережі, виконує та валідує транзакції користувачів, генерує нові блоки, надає зовнішній інтерфейс для взаємодії з клієнтськими додатками.

2. Android-додаток – клієнтське ПЗ для ОС Android, що використовує зовнішній програмний інтерфейс DLT вузла (API) для виконання різних транзакцій з аккаунтами користувача (повідомлення, платежі, голосування та ін.). Відповідає за реєстрацію та аутентифікацію користувача, зберігає його приватні ключі.

2.1.1.3 Загальні характеристики продукту

Системні інтерфейси

ПЗ DLT вузла запускається на операційних системах сімейства Linux, FreeBSD, Darwin (MacOS), Windows з встановленим середовищем виконання Java та GO.

Android-додаток запускається на операційній системі Android та відповідному емуляторі останніх версій.

Користувацькі інтерфейси

ПЗ DLT вузла надає інтерфейс командного рядка для конфігурування середовища запуску вузла та налаштування нового валідатора.

Android-додаток надає графічний інтерфейс користувача (GUI): форми з валідацією, підказки, головне меню, кнопки керування, індикатори прогресу виконання операцій, мультирядкові та однорядкові поля вводу.

Програмні інтерфейси

ПЗ DLT вузла надає метрики для моніторингу Spring Boot Admin та Prometheus через виділені RPC інтерфейси.

ПЗ DLT вузла надає REST API на базі формату даних JSON для відправки транзакцій, створення аккаунтів, керування вузлом та зчитування стану мережі, аккаунтів, вузлів-валідаторів та ін.

ПЗ DLT вузла реалізує та надає P2P API на базі amino кодування даних у вигляді gRPC сутностей.

Комунікаційні інтерфейси

ПЗ DLT вузла надає метрики для моніторингу через HTTP та JMX інтерфейс для Spring Boot Admin та Prometheus.

Комунікація між P2P вузлами забезпечується протоколом TCP з форвардингом портів на мережевих пристроях через UPnP.

Взаємодія між ПЗ DLT системи та клієнтськими додатками забезпечується через HTTP 1.1/2.0 для локальних клієнтів та HTTPS з TLS 1.3 для зовнішніх клієнтів підключених через мережу інтернет.

Обмеження пам’яті

Для ПЗ DLT системи кількість оперативної пам’яті необхідної для успішного запуску не повинно перевищувати 2 Гб, для функціонування під максимальним проектним навантаженням – не більше 4 Гб. Перевищення вказаних лімітів допускаються для сервісної чи аудиторської діяльності, для P2P рівня та підтримки консенсусу перевищення лімітів не допускається, для всіх інших операцій – лише короткостроково (не більше декількох хвилин) з обов’язковим збільшенням пропускної здатності та продуктивності додатку.

Android додаток запускається на пристроях з максимальним об’ємом доступної оперативної пам’яті в 512 Мб з подальшим зменшенням цього показника до 256 Мб у наступних версіях додатку.

Зовнішні операції

1. Підключення до P2P мережі з реалізацією сумісного протоколу відповідно до специфікації.

2. Налаштування генезисного блока з заданням аккаунтів з публічними ключами та балансами та початковими валідаторами (адреси, потужність, аккаунт для нагород.

3. Розповсюдження транзакцій серед підключених вузлів через широкомовний канал зв’язку P2P рівня.

4. Задання безумовного механізму хардфорків (оновлення критичних несумісних параметрів мережі) в зовнішньому конфігураційному файлі з застосуванням змін по висоті.

2.1.1.4 Функції продукту

Високорівневі функції DLT вузла є наступними:

1. Надання та реалізація P2P інтерфейсу комунікації з подібними вузлами.

2. Реалізація та підтримка dPOS консесусу.

3. Підтримка відправки транзакцій з та без комісійних зборів та з опціональним зашифрованим повідомленням.

4. Рейтингова система вузлів-валідаторів з штрафами для зловмисних (порушують консенсус) вузлів та нагородами для добросовісних (підтримують консенсус).

5. Випуск токенізованих активів з резервом у нативній валюті, можливість емітувати чи вилучати з обігу масу активу з пропорційним поповненням чи поверненням резерву.

6. Взаємне співіснування 3 типів аккаунтів: нативний, сумісний з bitcoin, сумісний з ethereum.

Високорівневі функції Android додатку змодельовані UML діаграмою варіантів використання, поданою на рис. А.1 та включають в себе:

1. Реєстрація та аутентифікація користувача.

2. Моніторинг стану DLT мережі (стан консенсусу, останній блок, непідтверджені транзакції, активні вузли і т.д.).

3. Відображення журналу транзакцій у вигляді бухгалтерської книги обліку.

4. Групування повідомлень по чатах та розшифрування їх вмісту при потребі.

5.Формування та попередня валідація транзакцій платежів, повідомлень, голосувань та їх офлайн підпис перед відправкою на DLT вузол.

6. Збереження користувацьких приватних даних (хешований пароль, приватні ключі).

7. Відображення токенізованих активів, якими володіє користувач та забезпечення операцій з ними (перекази, вилучення з обігу).

8. Відображення вузлів, за які користувач віддав свої голоси та можливість проголосувати за будь-який з доступних вузлів-валідаторів.

2.1.1.5 Характеристики користувачів

Користувачі ПЗ DLT вузла:

1. Адміністратор – відповідає за підтримку стабільної роботи вузла, проводить його первинне мережеве налаштування (відкриває зовнішні порти для P2P, назначає публічний ір або доменне ім’я, конфігурує інших пірів чи проксі для роботи з даним вузлом), налаштовує сервери моніторингу та оперативно реагує на аварійні та потенційно небезпечні ситуації, контролює кількість споживаних апаратних ресурсів та повідомляє розробників про знайдені помилки, виключні ситуації та дефекти, виявлені в процесі експлуатації.

2. DLT Розробник – відповідає за виправлення помилок та дефектів, додавання нових функцій та розробку технологічних рішень та архітектури системи в цілому, саме ПЗ DLT вузла використовує для налагодження нового коду, тестування мережевих сценаріїв використання (локальні мережі, архітектура sentry вузлів), конфігурування параметрів софт/хард форків і ініціації даних генезису.

3. Розробник клієнтських додатків – використовує REST API виділеного DLT вузла та документацію до нього для побудови власного клієнтського ПЗ на базі реалізованих доступних методів та відомих варіантів використання.

Користувачі Android-додатку поділяються на два класи і мають множину спільних можливостей.

1. Неавторизований користувач – може лише переглядати деякі дані про транзакції в DLT системі, не може відправляти транзакції, розшифровувати повідомлення.

2. Авторизований користувач – має доступ до всіх функцій додатку, може відправляти всі види транзакцій, переглядати та розшифровувати всі доступні чати.

2.1.1.6 Обмеження

Не допускається використання закритого або комерційного програмного забезпечення, а також бібліотек, готових модулів та вихідних кодів, що розповсюджуються з ліцензією на заборону комерційного та закритого використання продуктів на базі них, оскільки в подальшому DLT система може отримати комерційне застосування, а її програмний код може стати закритим в бізнес-інтересах.

Код Android додатку як і DLT вузла повинен бути написаний з використанням мови програмування Java, оскільки ця мова є однією з найбільш популярних у світі і має широке застосування у програмних продуктах рівня підприємства, а також є кросплатформенною, що дозволяє спростити сумісність додатків на різних операційних системах і винести накладні витрати на підтримку гетерогенних платформ на базі спільного ядра за межі даного проекту. Використання інших мов програмування повинно всебічно обґрунтовуватись, а доцільність їх застосування повинна бути доведена з приведенням у тому числі економічних ефектів та покращень, що будуть доступні при заміні чи частковій міграції мови програмування.

Для створення програмних інтерфейсів API повинна використовуватись технологія REST з форматом даних JSON, оскільки це спрощує інтеграцію Javascript-подібних клієнтів, дозволяє проводити швидкий аудит даних, що пересилаються між вузлами та є зручним для аналізу помилок та дефектів. Також даний формат є найбільш популярним у світі, тому обґрунтування інших рішень для програмних інтерфейсів повинно бути суттєвим та обов’язково доведеним.

2.1.1.7 Залежності та припущення

1. Криптографічні системи, що використовуються в bitcoin та ethereum стійкі до атак та не можуть бути скомпрометовані простим шляхом.

2. Google в подальших версіях SDK під Android буде підтримувати Java навіть при повному переході на розробку на Kotlin.

2.1.1.8 Абревіатури

ID - ідентифікатор
Dapos - ПЗ DLT вузла
Adap - Android додаток до ПЗ DLT вузла
GUI - графічний інтерфейс користувача
SRS - специфікація вимог до ПЗ
XML - розширювана мова розмітки

2.1.2. Вимоги

Даний розділ описує всі функціональні та нефункціональні вимоги, що пред’являються до розробки Dapos і Adap, а також відображає зовнішні інтерфейси між системами та визначає формати даних і процедури для них.

2.1.2.1 Зовнішні інтерфейси

Dapos повинен реалізовувати наступні зовнішні інтерфейси:

1. P2P – мережевий зовнішній інтерфейс, що повинен базуватися на TCP протоколі і відповідати за створення однорангової мережі з іншими вузлами, що мають сумісний протокол.

P2P повинен підтримувати форвардинг портів через UPnP для мережевих пристроїв, реалізовувати конфігуровані обмеження по часу отримання відповіді на кожен з типів запитів.

P2P повинен реалізовувати сумісність з самим собою (працювати в режимі клієнта та сервера одночасно).

P2P рівень повинен автоматично шукати та опитувати підключені вузли про нові вузли у мережі, доки не набере конфігурованої максимальної кількості одночасно підключених вузлів.

P2P повинен відключати вузли, що не відповідають вчасно або такі що порушують консенсус мережі.

Кожен вузол при підключенні до P2P іншого вузла повинен надіслати повідомлення про handshake та визначити криптографічні ключі для шифрування трафіку подібно до спрощеної моделі прийнятої у HTTPS.

2. REST API – клієнтський зовнішній інтерфейс, що повинен працювати через протокол HTTP 1.1/2.0 та HTTPS (TLS 1.3), і надавати доступ до відправлення транзакцій, керування вузлом, створення аккаунтів та зчитування стану різних сутностей з децентралізованого реєстру.

Документація до REST API повинна бути автоматично згенерована у форматі Open API Specification V3, клієнт до REST API повинен на базі даної специфікації бути доступним при запуску Dapos.

Авторизація для REST API не передбачається та забороняється (вузол не зберігає стан сесій користувачів та їх приватні дані).

2.1.2.2 Функції

Транзакції

1. Dapos повинен підтримувати можливість відправляти транзакції, що будуть змінювати стан децентралізованого реєстру.

2. Транзакції повинні містити адреси відправника, отримувача, суму переказу, сплачувану комісію та додаткові байти, що можуть використовуватись для службових потреб.

3. Тільки користувач що володіє ключем від аккаунта може відправляти транзакції від його імені.

4. Транзакції не можуть повторюватись, дублюватись чи пропускатись.

5. Транзакції при неавторизованій зміні повинні вважатися невалідними.

6. Транзакції повинні набувати алгоритмічної фінальності при потраплянні в блок.

7. Транзакцію, що підтвердилася в блоці, не можна відмінити.

8. Транзакції можуть виконувати довільні детерміновані зміни стану децентралізованого реєстру.

9. Транзакції, що потрапляють у блок можуть бути невалідними.

10. Невалідні транзакції не повинні призводити до будь-яких змін децентралізованого реєстру.

11. Транзакції після їх обробки можуть бути безпечно видалені з реєстру з збереженням всіх властивостей та вимог, яким транзакції повинні відповідати.

12. Транзакції можуть накопичуватися у пулі непідтверджених транзакцій, якщо Dapos не може їх підтвердити в поточному блоці.

13. Отримані Dapos транзакції розповсюджуються через P2P інтерфейс серед інших вузлів.

14. Порядок виконання транзакцій у блоці повинен бути детермінованим на всіх вузлах.

15. Валідність транзакції та результат її виконання не повинні залежати від недетермінованих факторів (системний час, випадкова величина та ін.).

Вузли

1. Будь-який комп’ютер, що має запущений Dapos та виділений Ip може стати вузлом.

2. Вузол повинен виконувати транзакції, розповсюджувати блоки та транзакції та підтримувати децентралізований реєстр у валідному стані.

3. Вузол повинен бути однозначно ідентифікований по ID.

4. ID вузла повинен використовуватись для створення захищеного P2P каналу.

5. Вузол може приймати участь у консенсусі, якщо за нього буде віддана достатня кількість голосів користувачів, що виражається у стейкінгу нативних токенів.

6. Вузол що приймає участь у консенсусі (далі валідатор) отримує нагороду за згенерований блок пропорційно ваги його голосів до загальної кількості голосів активних валідаторів.

7. Кількість валідаторів повинна бути обмежена, дане обмеження повинно змінюватись засобами безумовного хардфорку.

8. Список валідаторів повинен містити валідаторів, відсортованих по вазі голосів. Якщо валідаторів більше ніж місць у списку, то повинні обиратися перші n валідаторів по сумарній вазі голосів.

9. Валідатор може потрапити у список валідаторів лише наступним чином: є вільне місце у списку; він був доданий у список при створенні генезисного блока; він витісняє по вазі голосів існуючого валідатора.

10. Валідатор може втратити місце у списку валідаторів лише за наступних умов: він добровільно призупинив свої повноваження повідомивши детерміністичним способом про це інших валідаторів; його поведінка була розцінена більшістю вузлів як зловмисна; він перестав виконувати операції по підтримці консенсусу і після конфігурованої кількості попереджень був виключений з списку валідаторів; при виконанні хардфорку і скороченні місць у списку у нього не вистачило ваги голосів щоб поновитися у новому списку валідаторів; його витіснив з кінця списку новий валідатор, що отримав більше голосів.

11. Валідатор повинен бути оштрафований, якщо була виявлена зловмисна поведінка (візантійська відмова). Сума штрафу повинна конфігуруватись шляхом безумовних хардфорків або методом голосування валідаторів чи інших зацікавлених сторін.

12. Валідатор повинен бути оштрафований, якщо при сконфігурованих параметрах консенсусу він не в змозі своєчасно та в повному обсязі виконувати обов’язки по досягненню консенсусу, при цьому повинен існувати конфігурований період протягом якого валідатор може отримувати попередження, після досягнення максимальної кількості яких – він буде виключений зі списку та оштрафований на конфігурований відсоток стейку.

13. Валідатори повинні бути захищені від подвійних штрафів та їх повторення/накладання один на оден.

14. При візантійській відмові голоси валідатора повинні бути відкликані автоматично, а виборці повинні отримати зарезервовані депозити на свої рахунки.

Голосування

1. Кожен аккаунт, що має деякий сконфігурований об’єм депозиту повинен мати можливість проголосувати за будь-якого валідатора.

2. Аккаунт не повинен мати можливість голосувати за вузли, що не є валідаторами.

3. Вага голосу аккаунта за вузол визначається як сума нативних токенів мережі, що використовуються для такого голосування.

4. При голосуванні, аккаунт заморожує частину свого балансу, кожна одиниця нативних токенів може використовуватися для голосування лише один раз.

5. Аккаунт може у будь-який момент відкликати голос у валідатора, при відкликанні аккаунт отримує на рахунок заморожену суму нативних токенів і може ними вільно розпоряджатися.

6. Аккаунт може голосувати за довільну кількість валідаторів пропорційно розподіляючи вагу голосів.

7. У кожного валідатора є сконфігурована консенсусом кількість доступних слотів для голосування.

8. Аккаунт може витіснити інший аккаунт з списку слотів для голосування, у випадку якщо всі слоти є заповненими, проте вага голосу даного аккаунту більша за мінімальну вагу голосу в даному списку. Для витісненого аккаунту голос повинен бути автоматично відкликаний.

9. Аккаунт може укрупнювати власні голоси у слотах валідаторів голосуючи декілька разів за одних і тих самих валідаторів.

10. Якщо валідатор, за який проголосував аккаунт був оштрафований, то такий аккаунт відповідно до ваги свого голосу несе солідарні втрати замороженого балансу.

11. Валідатори не можуть голосувати один за одного або за себе.

Консенсус

1. Відсоток участі (відносна сила) вузла у консенсусі повинен підтримуватись на базі відношення кількості голосів окремого валідатора до загальної кількості голосів – dPOS.

2. Параметри консенсусу повинні задаватися механізмом безумовних хард/софтфорків.

3. Консенсус повинен бути досяжним навіть при візантійській відмові до 33% валідаторів по вазі голосів.

4. Консенсус повинен бути синхронним, створення форків та побудова додаткових ланцюжків не допускається.

5. Участь валідаторів у побудові конкурентних ланцюжків повинна бути заборонена, а у разі її виявлення – валідатор повинен бути оштрафований.

Повідомлення

1. Кожна транзакція повинна мати можливість додавання опціонального повідомлення.

2. Повідомлення повинно бути зашифрованим.

3. Для шифрування повідомлення повинна використовуватись схема Діффі-Хелмана для генерації симетричного ключа.

4. Для шифрування симетричним ключем повинен використовуватись алгоритм AES в режимі GCM без відступів з одноразовим випадковим числом nonce для модифікації вихідного шифротексту, щоб уникнути подібності шифротекстів.

5. Симетричний ключ повинен генеруватись, використовуючи ключі відправника та одержувача (повідомлення між двома учасниками), також опціонально користувач може провести генерацію лише своїми ключами (анонімні дані).

6. Повідомлення можуть бути розшифровані користувачем шляхом відновлення симетричного ключа за допомогою схеми узгодження ключів Діффі-Хелмана.

Сховище ключів

1. Аккаунти користувачів можуть зберігатися в Dapos, проте рекомендованим є зберігання у Adap.

2. Сховище ключів повинно зберігати приватні ключі аккаунтів, що йому довіряють таке право, в зашифрованому вигляді.

3. Шифрування ключів повинно використовувати: алгоритм симетричного шифрування AES-128 режим GCM без відступів, функцію хешування SHA-256, одноразові псевдовипадкові числа nonce.

4. Шифрування ключів однією і тією ж парольною фразою повинно приводити до різного вихідного шифротексту.

5. Сховище повинно дозволяти однозначно визначити – чи валідна парольна фраза використана для його розкриття.

6. Парольна фраза не повинні ніде зберігатись, нею володіє лише користувач.

2.1.2.3 Вимоги до продуктивності

Час розгортання Dapos повинен бути менше 10с.

Час підключення до вузлів P2P мережі повинен бути менше 30с.

Швидкість обробки транзакцій повинна бути не менше 1000 в секунду при 20 вузлах в мережі з 4Гб пам’яті та 4 ядерними процесорами з тактовою частотою більше 4 Ггц.

Час створення нового блоку не повинен перевищувати 6 с.

Dapos повинен підтримувати розмір реєстру до 4 Тб.

Час виконання простого запиту на читання з бази даних повинен бути менше 3с, при читанні діапазона з пагінацією чи фільтруванням – не більше 15 с., при оновленні даних чи додавання нових – не більше 5с.

Час підтвердження транзакції не повинен перевищувати 12 с., якщо пул непідтверджених транзакцій не повністю заповнений.

Час запуску Adap на пристрої зі Snapdragon 650 та 3 Гб пам’яті не повинен займати більше 5с.

2.1.2.4 Обмеження проектування

Не дозволяється використання примітивів шифрування крім таких, що вказані в специфікації.

Заборонено впроваджувати для Dapos використання сторонніх систем при досягненні консенсусу

Заборонено використовувати принципи централізованого проектування при розробці архітектури Dapos.

Заборонено зберігати персональні дані користувача у відкритому вигляді.

Не дозволяється використання консенсусів, що підтримують форки та механізм найдовшого ланцюжка.

2.1.2.5 Атрибути програмної системи

Dapos повинен запускатись у програмному середовищі Java 11 та Go 1.14 на останніх версіях Linux, Darwin, Windows, FreeBSD.

Adap повинен запускатись на операційні системі Андроїд версії 6 та новіше.

Adap та Dapos повинні очищати секретні дані користувача відразу після їх використання.

2.1.3. Верифікація

Верифікація повинна проводитись за допомогою CI/CD пайплайнів з використанням модульних та інтеграційних тестів.

2.2. Проектування

Відповідно до поданих у підрозділі 2.1 вимог необхідно розробити архітектуру програмного забезпечення системи забезпечення довіри.

Весь процес проектування і побудови майбутнього ПЗ системи забезпечення довіри в децентралізованих системах подано у формі документу SAD (Software Architecture Document) в додатку Б.

В даному підрозділі будуть описані лише основні проектні рішення системи, а їх деталізація і низькорівневий опис, в тому числі і рисунки та діаграми будуть подані у SAD.

Згідно з принципом поділу архітектури програмного забезпечення “4+1”, архітектура складається з наступних рівнів:

1. Логічне представлення – спосіб організації коду, класи пакети, зв’язки між ними. Для розроблюваної системи була обрана шестигранна архітектура, що дозволяє розділити код системи подібно до багатошарової архітектури на бізнес логіку та зовнішні вхідні та вихідні порти, що надають чи потребують реалізації певного інтерфейсу взаємодії (веб-контролери, енд-поїнти, користувацький інтерфейс, доступ до бази даних, доступ до P2P підсистеми, доступ до рушія консенсусу). Перевагою цієї архітектури над багатошаровою є можливість побудови системи навколо бізнес-логіки з довільною кількістю зовнішніх інтерфейсів, що з

нею взаємодіють, не дотримуючись формального поділу на шари, що для невеликих додатків створює більше більше плутанини та зайвого коду та змушує розробників писати процедурний код на об’єктно-орієнтованій мові програмування. Шестигранна архітектура також дозволяє просто ізолювати бізнес-логіку від зовнішніх сутностей та провести її модульне тестування та інтеграційне тестування з штучними зовнішніми елементами. Такий підхід дозволяє сфокусуватись на розробці важливих для додатку функцій та проводити їх оперативне впровадження без необхідності проводити тестування всієї системи в цілому. Шестигранна архітектура – є прийнятим стандартом для мікросервісної архітектури, що ще раз доводить її ефективність для невеликих додатків.

2. Представлення реалізації – результат виконання збірки проекту. Таке представлення складається з модулів, що представляють скомпільований і зібраний код, і компонентів, які є виконуваними чи такими, що розгортаються і містять один або декілька можулів. Для Java модуль, як виконуваний, так і залежний мають представлення Zip архіву з розширенням .jar або .war (для модуля,що розгортається на веб-сервері), що містить скомпільовані файли вихідного коду, розбиті на пакети, а також маніфест з мета-інформацією. Зв’язок між модулями визначається залежностями та тим, які компоненти об’єднані у той чи інший модуль. Для цього рівня представлення існує два архітектурних стиля – монолітний та мікросервісний. Кожен з цих стилів обирається під свої унікальні задачі, проте як правило більшість проектів на початкових етапах існування обирають монолітну архітектуру в силу її простоти організації, підтримки і супроводу при невеликій кодовій базі та кількості функцій продукту. Є відомий архітектурний шаблон, що називається Monolith First, що визначає монолітну архітектуру, як єдину можливу першу архітектуру представлення для нового проекту. По мірі розростання проекту, архітектурний стиль може бути змінений на мікросервісний, тому даний підхід і буде обраним для організації архітектури представлення реалізації для ПЗ DLT вузла. Кінцевим артефактом, що буде розгортатися буде fat jar.

3.Представлення процесу – позначає взаємодію компонентів на етапі виконання з точки зору системного інженера і позначається як правило UML

діаграмами станів, діяльності, компонентів та послідовності. Основним елементом на даному рівні виступає процес, що виконується, а елемент що їх об’єднує – представляється механізмом взаємодії між процесами.

4. Представлення розгортання – визначає способи розподілу компонентів між фізичними пристроями. Елементи в даному представленні складаються з серверів (фізичних та віртуальних) і процесів. Зв’язки між серверами представляють мережу. Це представлення додатково описує відношення між процесами та пристроями. У випадку розробки DLT системи архітектура типу представлення розгортання може бути охарактеризована як клієнт-серверна, оскільки згідно поданої специфікації вимог (SRS) необхідно розробити ПЗ вузла DLT системи (сервер) та клієнтський Андроїд додаток (клієнт).

5. Сценарії – історії користувачів (User story) та варіанти використання сукупності функцій ПЗ, що пов’язують всі 4 види представлення архітектури у єдине ціле створюючи модель представлення архітектури “4+1”. Сценарії як правило подаються діаграмами діяльності та варіантів використання UML, що для розроблюваного ПЗ подана на рис. А.1.

У ході розробки архітектури в рамках представлення “4+1” було створено високорівневу діаграму компонентів UML для опису системи в цілому та відображення основних комунікаційних інтерфейсів між компонентами.

Компоненти в даному випадку представляють декомпозицію системи на функціонально незалежні частини, кожна з яких відповідає лише за обмежений перелік пов’язаних операцій (Single responsibility) – перший принцип практик SOLID. Сама діаграма компонентів зображена на рис.2.1.

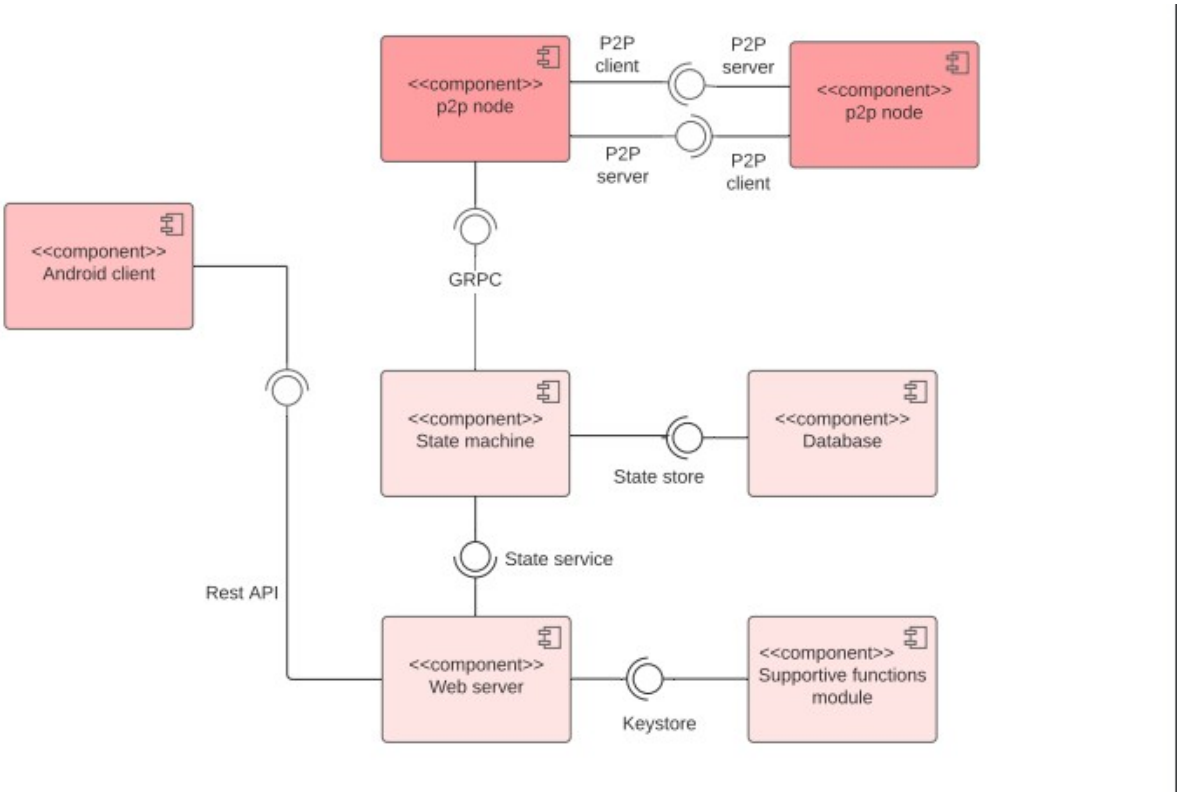


Рис 2.1. Високорівнева діаграма компонентів UML

Згідно діаграми компонентів архітектура системи будується навколо компоненту State machine, що визначає детермінований скінчений автомат DLT вузла, який відповідає за валідацію і виконання транзакцій в мережі. Вузол реалізовує сервіси зчитування даних поточного стану автомата (кінцеві дані, що зберігаються в базі даних), State machine відповідає за їх форматування та організацію таким чином, щоб їх міг прочитати веб сервер через простий та зрозумілий сервісний інтерфейс (State Service). Сам State Service є вхідним портом із зовнішнього світу користувачів DLT системи до її бізнес логіки – State Machine. Такий підхід дозволяє ізольовано тестувати скінчений автомат без необхідності розробки та розгортання клієнтських додатків, також це дозволить швидше та точніше виявити наявний дефект. Web server представлений як окремий компонент, оскільки його основна задача реалізовувати REST API, що вимагається в специфікації вимог, також веб сервер потребує взаємодії не лише зі станом DLT мережі через інтерфейс скінченого автомата, а й виклику адміністративних та додаткових функцій по маніпулюванню локальними налаштуваннями вузла,

моніторингу стану вузла (пам’ять, завантаженість процесорних ядер), створення аккаунтів в сховищі ключів і т.д. Всі ці функції винесені в окремий компонент – Supportive functions module, що агрегує доступ до всіх додаткових, непов’язаних з децентралізованим реєстром можливостей. Розбиття на такі компоненти дозволяє у майбутньому зібрати артефакт, що не містить веб-сервера взагалі, як і модуля додаткових функцій, що дозволить зменшити кількість споживаних ресурсів апаратного забезпечення та розмір кінцевих пакетів дистрибуції.

Андроїд клієнт зображений як окремий компонент, що працює з ПЗ DLT вузла через програмний інтерфейс – Rest API. API у свою чергу реалізується та надається веб-сервером, що дозволяє згенерувати заглушки для API сервера та проводити тестування Андроїд клієнта в штучному, ізольованому середовищі.

Скінчений автомат для функціонування потребує реалізації інтерфейсу State Store (сховище стану), цей інтерфейс використовується компонентом State machine для зчитування поточного стану та виконання його детерміністичних змін при виконанні транзакцій (здійснення state transition). Використання окремого інтерфейсу для збереження стану виправдано тим, що таким чином ізолюється логіки обробки транзакцій від логіки збереження та зчитування, також така декомпозиція дозволяє змінювати реалізації State Store без необхідності повного переписування програмного коду.

Компонент Database в даному випадку виступає сховищем даних, на який накладається реалізація State Store, база даних може бути будь-якою, проте при її виборі варто зважати на вимоги по продуктивності системи та її незалежність від закритих і комерційних технологій, що приводяться у SRS.

Скінчений автомат через gRPC інтерфейс взаємодіє з p2p вузлом, gRPC інтерфейс є високопродуктивним способом передачі серіалізованих у бінарному вигляді, версійних сутностей даних між процесами, що працюють на різних мовах програмування. В такому випадку p2p вузол реалізовує базовий p2p консенсус в мережі (наприклад PBFT, RBFT) та потребує від скінченого автомату реалізації функцій по доставці та валідації транзакцій, а також реалізації консенсусу другого рівня, такого як POS, dPOS або ін.

P2P вузол є одночасно як клієнтом так і сервером у DLT мережі та виконує сканування інших доступних P2P вузлів з сумісним протоколом. P2P вузол відповідає за формування P2P мережі у якій буде досягатись консенсус першого рівня, будуть розповсюджуватись транзакції, блоки, голоси консенсусу, докази зловмисної поведінки і т.і. Особливістю компонента P2P вузла є можливість тестування без необхідності створення мережі, замість цього розробляється простий сумісний клієнт-сервер, що буде взаємодіяти з клієнтом та сервером P2P вузла. Таким чином ізольовано перевіряються функції, реалізовані на мережевому рівні.

Процедура відправки транзакції платежу зображена на діаграмі послідовності UML на рис.2.2.

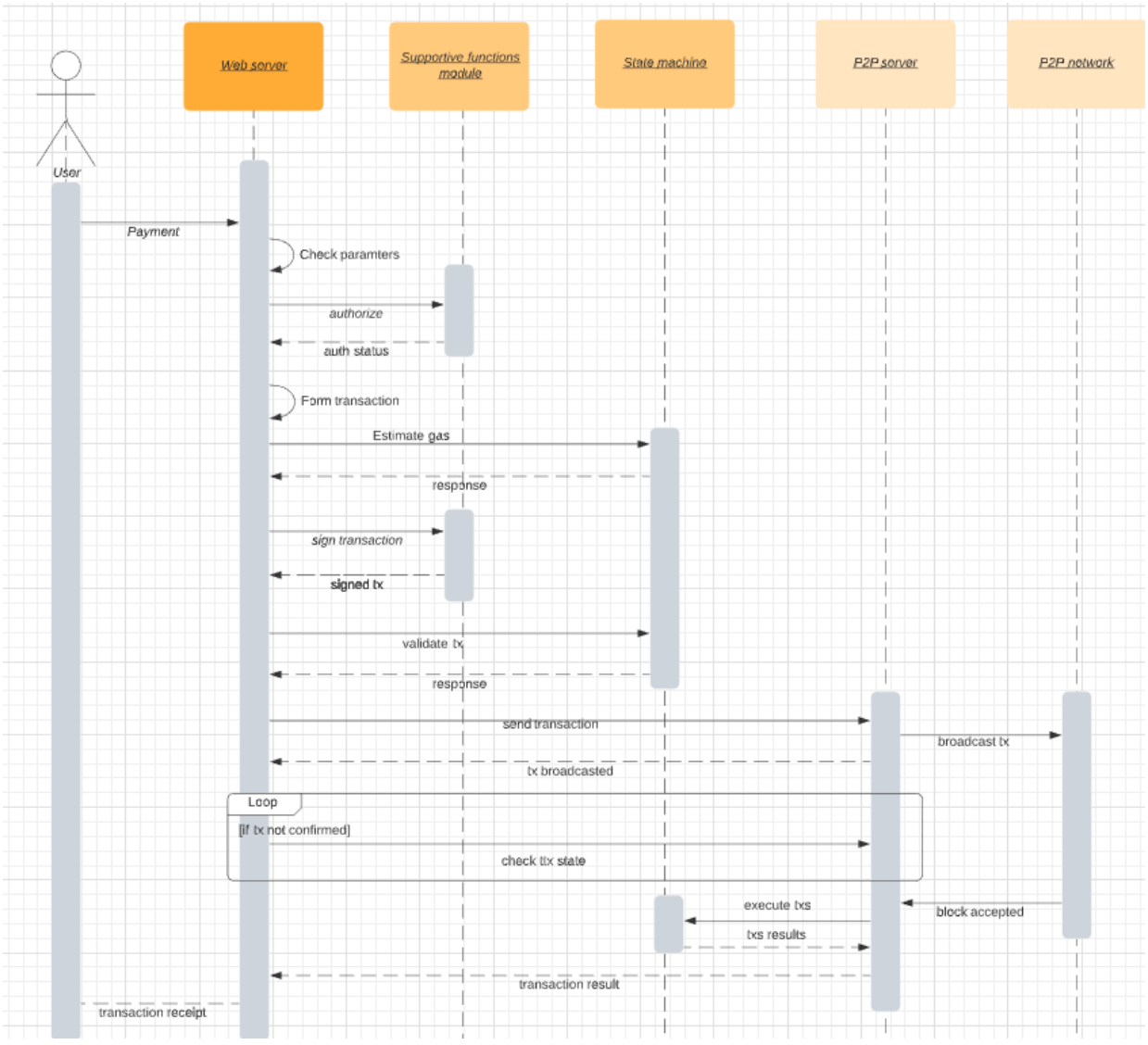


Рис 2.2. Діаграма послідовності UML для відправки транзакції

В даному випадку упущений Android клієнт, оскільки Rest API може використовуватись напряму через генерований по Open API Specification V3 клієнт. Користувач виступає ініціатором платежу, вказуючи свою парольну фразу, адресу аккаунта, з якого він хоче відправити кошти, отримувача, суму, додаткове повідомлення та параметри його шифрування (ключі що будуть використовуватись), веб сервер отримує запит користувача та перевіряє всі параметри на валідність. Після проведення такої перевірки веб-сервер намагається авторизувати користувача шляхом перевірки правильності вказаної парольної фрази через компонент сховища ключів що входить до модулю додаткових функцій, якщо користувач вказав пароль та ID аккаунту вірно, то веб-сервер формує тіло транзакції, що необхідно відправити і оцінює кількість затраченого газу (абстрактної величини, що позначає складність виконання операції) на таку транзакцію викликаючи компонент State Machine. Після отримання параметру кількості газу, веб-сервер присвоює це значення транзакції як максимально можливу кількість газу для її виконання і виставляє ціну газу прийняту за середню в поточний час для мережі. Далі веб-сервер викликає авторизоване сховище ключів для проведення підпису транзакції від імені користувача. Підпис виконується для того, щоб транзакції міг проводити лише власник приватного ключа до аккаунта, а підробка чи зміна контенту транзакції (в тому числі підпису) приводила до невалідності транзакції.

Підписана транзакція відправляється на P2P вузол, де вона потрапляє у пул непідтверджених транзакцій, які вибираються недетермінованим способом для підтвердження при створенні блоку. Всі транзакції перед потраплянням в пул обов’язково проходять повторну валідацію на скінченому автоматі, що аналогічна, тій що проводив веб-сервер перед її відправленням, проте також така валідація може бути легковісною та не виконувати операцій, що потребують зчитування стану кінцевого автомата.

Підписана транзакція після потрапляння в пул – розповсюджується серед інших вузлів методом відправки найближчим n сусідам (найширший канал, найменша затримка відповіді). До тих пір, поки транзакція є в пулі, вона розповсюджується серед інших вузлів. Про те, що транзакція була передана в

мережу, P2P вузол повідомляє веб-сервер. Веб-сервер з цього моменту починає опитувати P2P вузол в циклі з певним періодом (наприклад 3с) про стан транзакції (прийнята/відхилена, підтверджена/непідтверджена). Доки транзакція не є підтвердженою (у випадку незаповненого пулу непідтверджених транзакцій цей час повинен складати до 2 періодів генерації блоків), веб-сервер очікує її підтвердження. Коли P2P мережа приймає новий блок, то P2P вузол сканує всі транзакції цього блоку і запускає їх виконання на скінченому автоматі (State Machine), результати транзакцій P2P вузол записує в реєстр та надає інформацію веб-серверу про відправлену ним транзакцію, якщо вона була прийнята в блоці. Результат виконання транзакції передається як відповідь користувачу.

2.3. Обгрунтування методології і засобів розробки

Для реалізації програмної системи описаної у попередньому підрозділі буде використана мова програмування Java, оскільки ця мова програмування дозволяє розроблювати кросплатформені додатки для різних операційних систем та архітектур комп’ютерів, а також на відміну від C++ має кращий рівень безпеки і автоматизує чи делегує рутинні операції з пам’яттю віртуальній машині, що значно знижує кількість можливих помилок та підвищує тестованість і швидкість розробки додатків.

Основною причиною вибору Java є висока портованість і сумісність, що дозволить досягти високого рівня доступності ПЗ на різних платформах, маючи при цьому лише один екземпляр скомпільованого коду.

Також ще однією причиною вибору мови програмування Java є велика екосистема сторонніх проектів, фреймворків та бібліотек у відкритому доступі, що дозволить реалізувати кінцевий програмний продукт будь-якої складності за відносно невеликий час.

Останньою причиною використання Java є наявність професійного досвіду розробки на цій мові програмування блокчейн системи Apollo у автора даної

магістерської роботи, що дозволить більше сфокусуватись на досягненні цілей, а не на вивченні можливостей бібліотек і фреймворків.

Для реалізації основних задач було обрано наступні програмні засоби:

1. IntelliJ IDEA – інтегроване середовище розробки ПЗ мовою програмування Java, використовується для кодування, налаштування та тестування ПЗ;

2. Jetbrains Xodus – високопродуктивна NoSQL база даних з рівнем ізоляції “snapshot”, кращою швидкістю випадкового читання і запису ніж у вбудованих SQL аналогів (H2, Apache Derby, HSQLDB) і підтримкою key-value та entity сховищ;

3. Spring Boot – фреймворк, що автоматизує створення додатку на базі Spring Framework і забезпечує автоматичне розгортання веб-серверу сервлетів, контейнеру інверсії залежностей, конфігурування безпеки і журналювання, підключення баз даних та проксювання компонентів для підтримки AOP.

4. Apache Tomcat – веб-сервер для надання REST API для можливості відправки транзакцій через клієнтський додаток.

5. Gradle – інструмент для збирання проекту, запуску модульних тестів, підключення залежностей і генерації артефактів для спрощеного запуску і дистрибуції ПЗ;

6. Git – система контролю версій, використовується для збереження даних проекту, відновлення втрачених фрагментів коду та ведення історії версій ПЗ.

7. CircleCI – хмарна система безперервної інтеграції, що запускає збірку проекту у хмарі при кожній зміні стану репозиторію і дозволяє підключати сторонні сервіси та задачі формуючи пайплайн.

8. SonarCloud – хмарна система інтелектуального аналізу програмного коду, що оцінює кількість можливих дефектів, ризики безпеки, порушення конвенцій програмування та показує, скільки часу необхідно на їх усунення.

9. Jacoco – плагін для збірки проекту Gradle, що генерує звіт з покриття коду тестами (рядки, класи, пакети, гілки).

10. GitHub – хмарний провайдер репозиторіїв Git, що дозволяє підключати до проектів хмарні сервіси та відслідковувати статус проекту (збірка, покриття тестами, якість коду).

2.4. Висновки до розділу

У ході підготовки до розробки програмного забезпечення було розроблено специфікацію вимог SRS, що базується на колі проблемних питань до вирішення, що представлені в підрозділі 1.5. Специфікація вимог включає в себе функціональні та нефункціональні вимоги, вимоги до продуктивності, безпеки та надійності, а також визначає обмеження розробки продукту, його проектування та використання відомих залежностей чи зовнішніх компонентів. Опис зовнішніх інтерфейсів допоміг при розробці проекту програмного забезпечення, оскільки явно було визначено взаємодію кожного компонента програмної системи із зовнішнім світом, проведено функціональну декомпозицію та опис варіантів використання, що дозволить в подальшому ефективно реалізувати можливості і характеристики ПЗ.

При розробці архітектури, було обрано шестигранну логічну архітектуру (для спрощення тестування бізнес-логіки), монолітну архітектуру представлення реалізації (легке розгортання та підтримка невеликих проектів) і клієнт-серверну архітектуру розгортання (P2P мережа, Android додаток та Rest API сервер ПЗ DLT вузла), що дозволяє організувати код, компоненти, артефакти і процеси найбільш ефективним способом, що дозволить у кінцевому випадку пришвидшити розробку, тестування і розгортання DLT мережі на базі ПЗ, розроблюваного в рамках даної магістерської роботи.

При виборі технічних рішень увага приділялась сумісності з вимогами, вказаними у специфікації, тому основною мовою програмування було обрано Java, доведено доцільність цього вибору (кросплатформеність, широкий вибір бібліотек, надійність та популярність), також відповідно до мови програмування було обрано набір фреймворків, основними з яких є Spring Boot та Tendermint Core, що дозволяють у швидкі терміни побудувати децентралізовану систему з вбудованим і готовим до використання API сервером, реплікацією реалізованого скінченого автомата та P2P рівнем взаємодії, що ізольований від інших компонентів та не впливає на консенсус другого рівня.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Розробка ПЗ

Задачею цього підрозділу є опис процедури розробки DLT системи в цілому, подання застосованих алгоритмів та прийнятих низькорівневих рішень, що забезпечують підтримку розроблених у розділі 2 вимог, архітектури та приведених компонентів.

ПЗ DLT вузла являє собою комбінацію трьох компонентів, що представлені двома окремими процесами:

Перший процес – P2P вузол, що базується на фреймворку Tendermint Core, в його основні задачі входить:

1. Забезпечення довіри і механізмів репутації для підключених по P2P вузлів (TS компонент)

2. Розповсюдження транзакцій і т.і,

3. Виклик через GRPC інтерфейс АПІ реплікованого скінченого автомата для підтримки консенсусу та додавання і валідації транзакцій.

Другий процес – представлений виконуваним середовищем Java і побудований на базі фреймворку Spring Boot та включає в себе наступні компоненти:

1. Веб сервер для реалізації REST API.

2. Скінчений автомат для консенсусу другого рівня та виконання транзакцій.

3. Шар доступу до даних (Dao) на базі NoSql бази даних Jetbrains Xodus

4. Модуль додаткових функцій підтримки: сховище ключів, сервіс, підписуючий транзакції, сервіс формування транзакцій.

Контекстна діаграма класів і пакетів зображена на рис.3.1. Ця діаграма зображує структурну організацію пакетів додатку та дає представлення про логічну архітектуру, що прийнята при розробці.

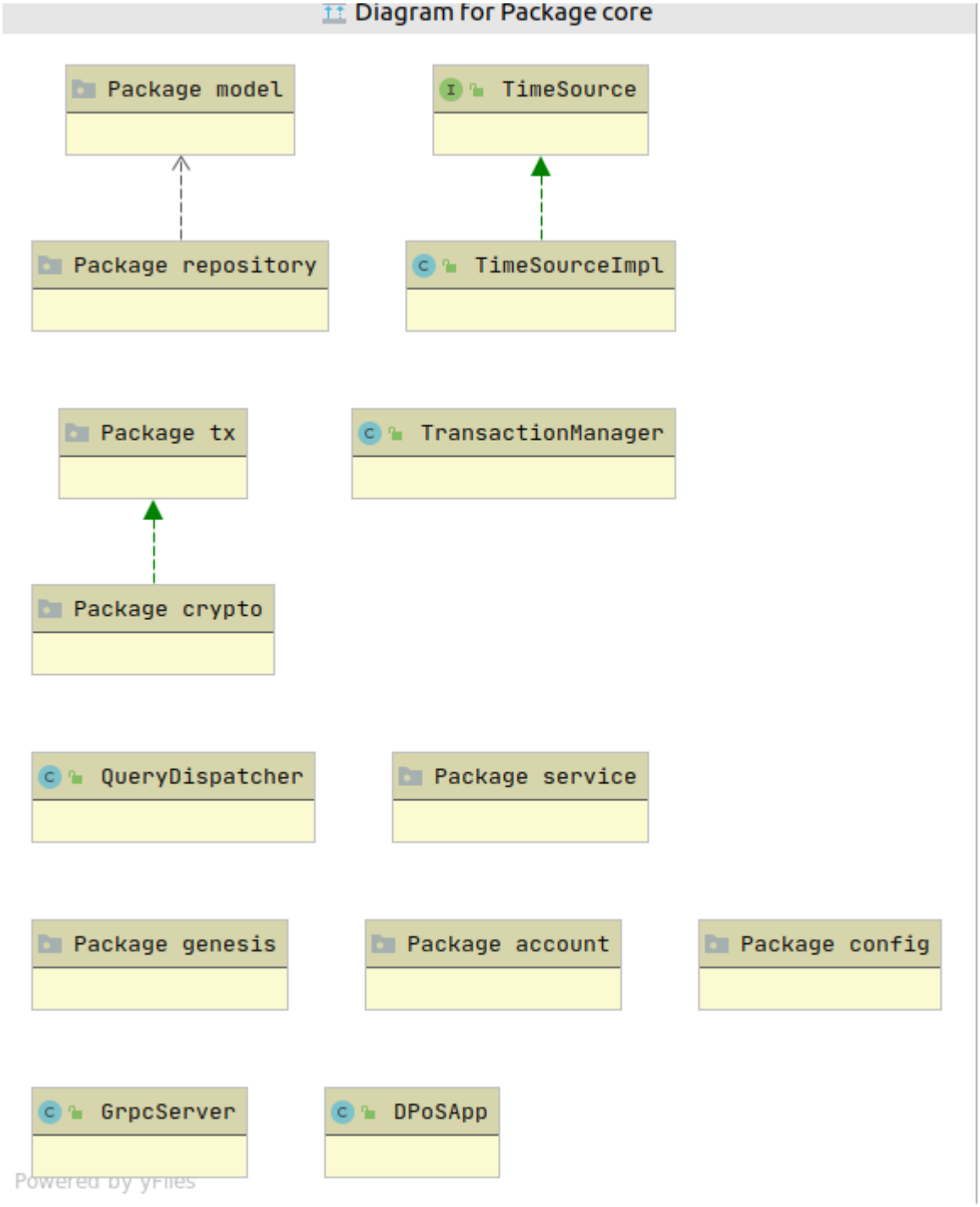


Рис 3.1. Діаграма класів і пакетів

У ході розробки для документування Rest Api було використано Spring Fox та Swagger annotations що дозволили згенерувати документацію про доступні функції для виклику у вигляді OpenApi Specification V3 та розгорнути згенерований на базі клієнт в якості першої реалізації Http клієнта, що зображений на рис.3.2.

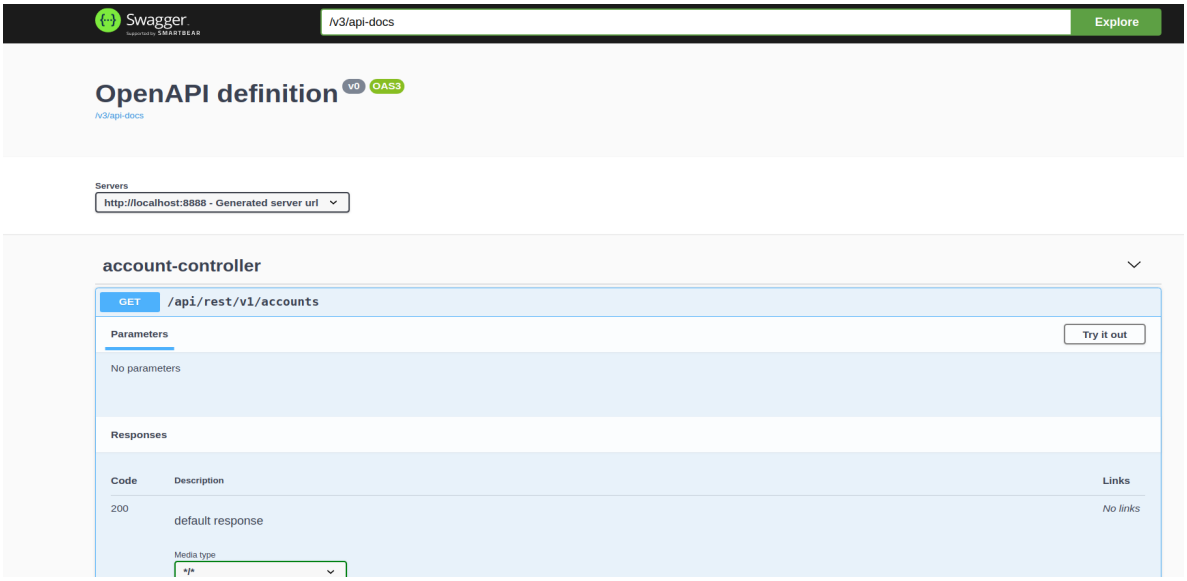


Рис 3.2. Swagger клієнт до Rest API

3.2. Демонстрація можливостей DLT системи

Демонстрація запущеного додатку Android-додатку приведена на рис.3.3.

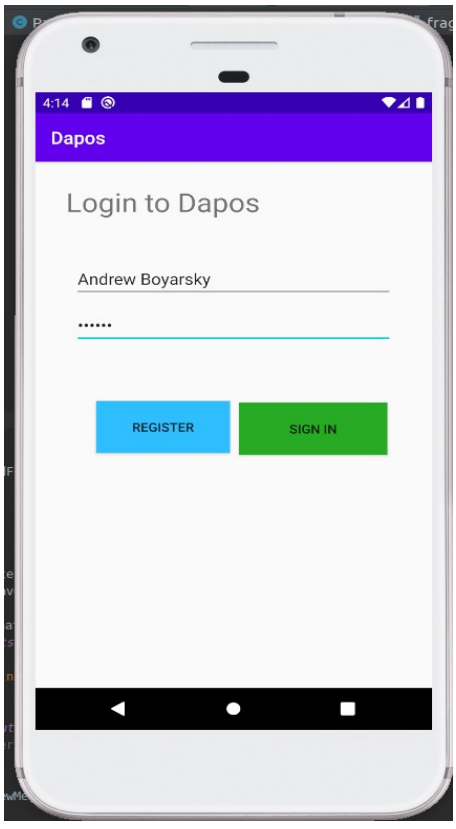


Рис 3.3. Запущений Андроїд додаток

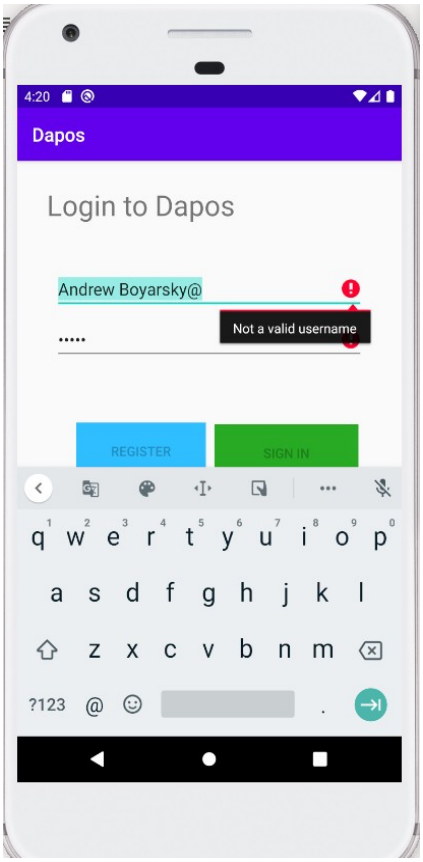


Рис 3.4. Помилки вводу

При правильному введенні даних користувача або реєстрації нового, створюється новий гаманець з криптографічними ключами, а користувача переводить на екран перевірки балансу аккаунту, що зображений на рис.3.5.

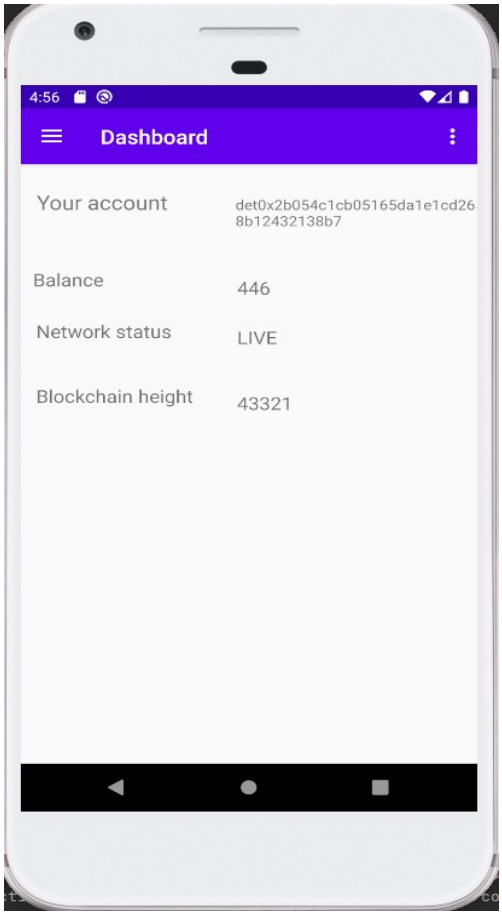


Рис. 3.5 Екран перевірки балансу

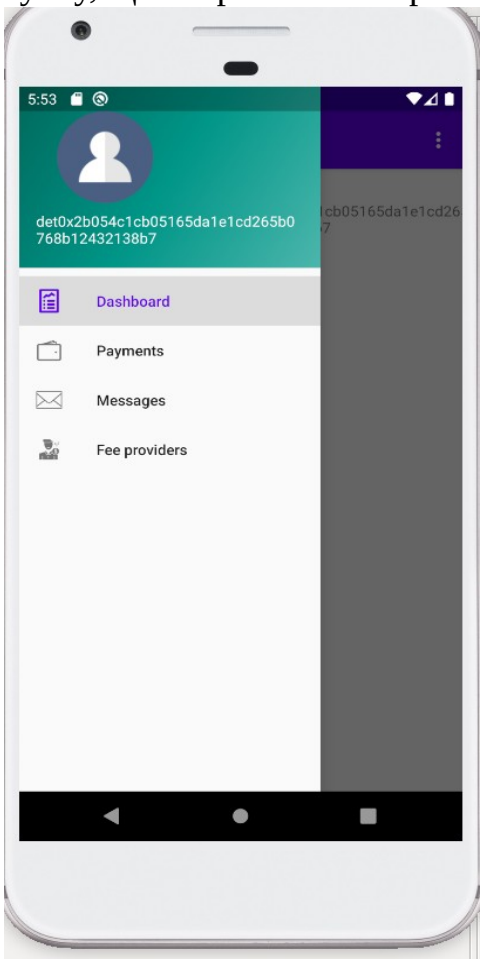


Рис.3.6 Головне менюДля відправки транзакцій користувач може використовувати головне меню що відкривається з кнопки з верхнього лівого куту екрана, що зображене на рис.3.6. Користувач може обирати з доступних функцій відправки платежу, повідомлення чи створення власного платника послуг.

Далі буде описано процедуру відправки платежу, з використанням автоматичного платника послуг та без комісії, відповідна процедура починається з меню Payments, натиснувши на яке користувач переходить до списку транзакцій платежів, в цьому ж меню зображена кнопка створення нового платежу, що дозволяє відправити новий платіж у процедурі що зображена на рис.3.7.-.3.9. Для відправки необхідно ввести отримувача, суму та шифроване повідомлення за бажанням.

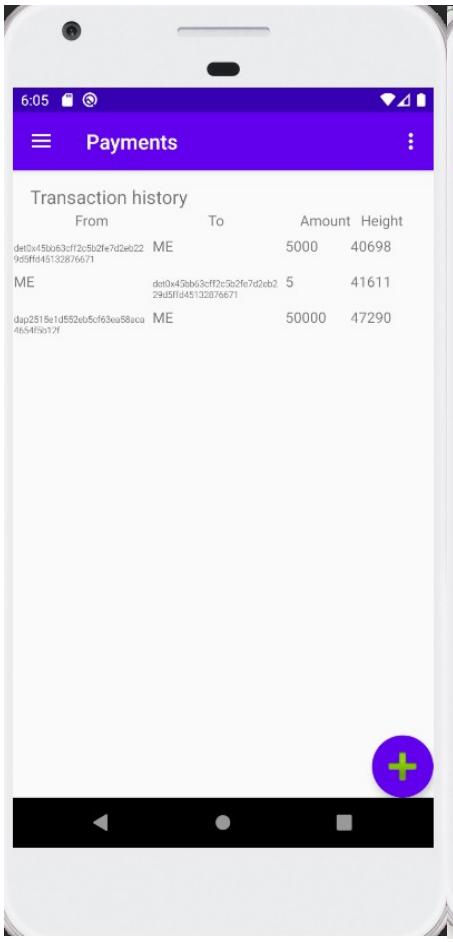


Рис.3.7. Платежі

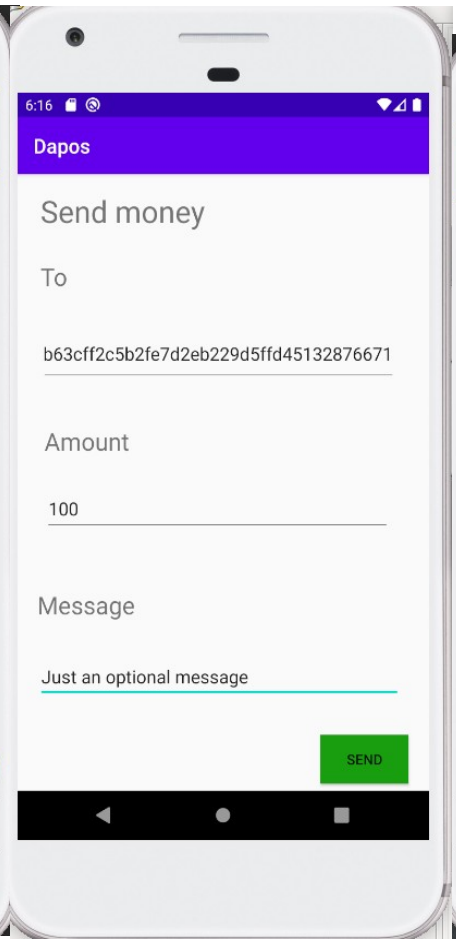


Рис.3.8. Новий платіж

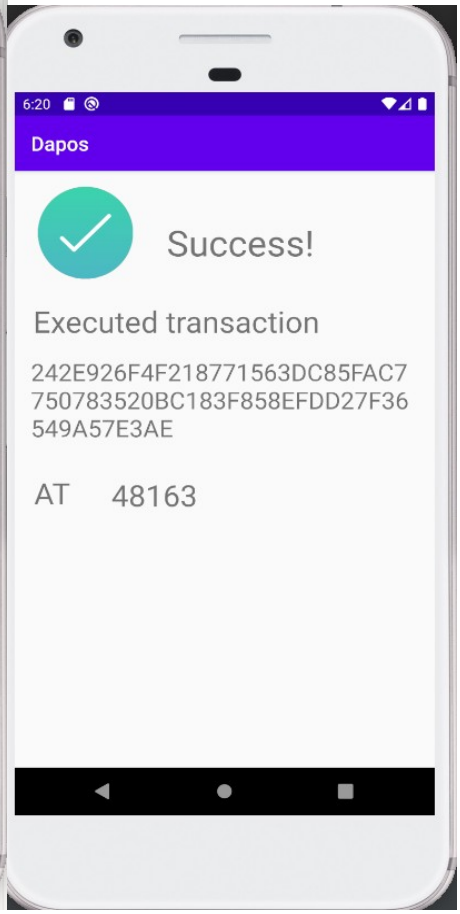


Рис.3.9. Підтвердження

Кожен платіж потребує підтвердження від DLT мережі тому клієнт при виконанні запиту очікує доки транзакція буде прийнята мережею шляхом прямого блокування виклику Rest API.

Виконувати операції з DLT системою можна і через Swagger UI, що дозволяє швидко протестувати необхідну функцію чи цілий сценарій без необхідності мати виділений клієнт (як андроїд, що описувалось вище).

Так відправлення транзакції через Swagger клієнт з підтримкою fee provider показано на рис.3.10. В даному випадку використовується доступ до розробленого REST API на ендпоїнт якого відправляється сформований запит у вигляді JSON, де вказується відправник, сума, отримувач, пароль до аккаунта, постачальник послуг оплати трансферу і опціональне повідомлення. Після натискання кнопки Execute, запит буде виконано, а відповідь буде отримана після успішного виконання операції в децентралізованій мережі з її повною верифікацією.

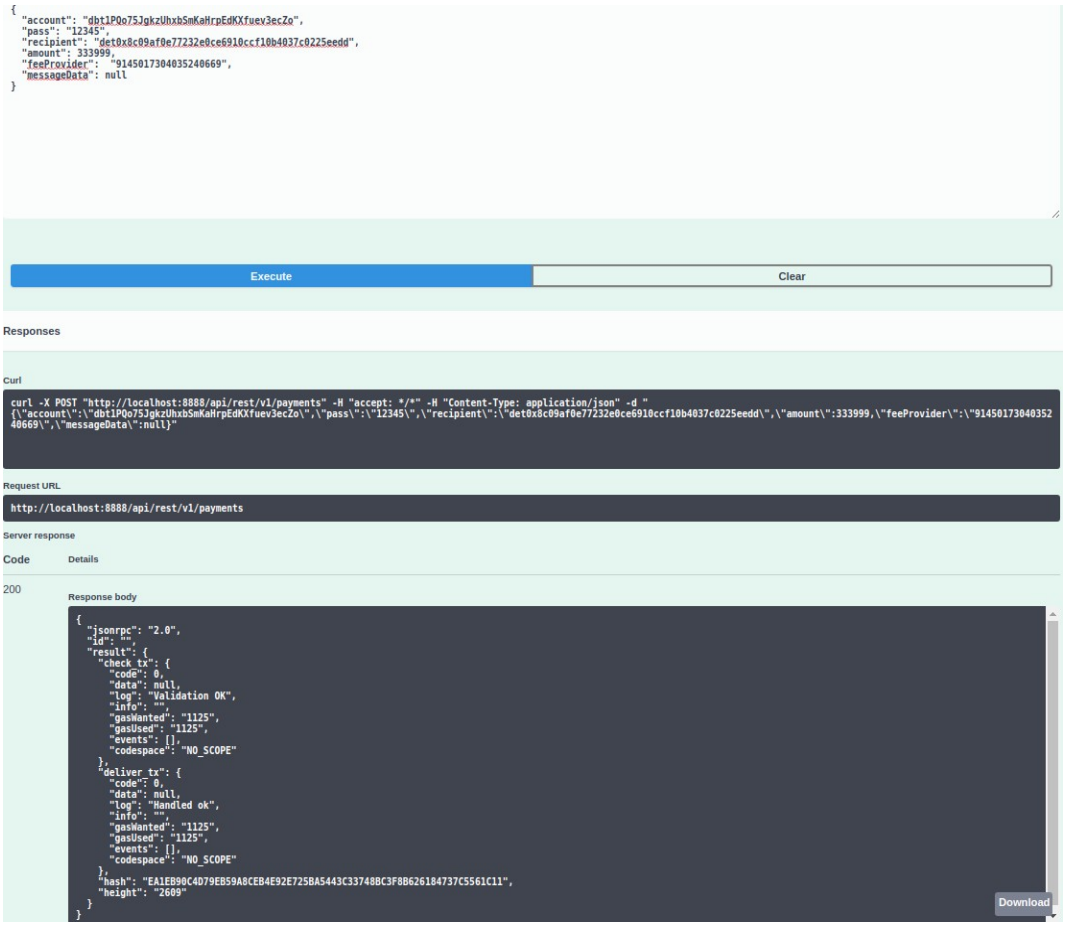


Рис.3.10. Відправка транзакцій через Swagger

3.3. Якість ПЗ

Після реалізації комплексу програмного забезпечення у рамках даної роботи було проведено оцінку його якості, а саме вихідного коду шляхом аналізу статичних метрик засобами Sonar Cloud, результати цього аналізу зображені на рис.3.10.

Як видно з проаналізованих метрик надійності, підтримуваності, потенційних помилок і технічного боргу, розроблена система отримала статус A — найвищий ступінь якості програмного забезпечення, що відповідає тому, що під час розробки вихідного коду використовувались кращі практики об’єктно-орієнтованого програмування з дотриманням правил “чистого коду”.

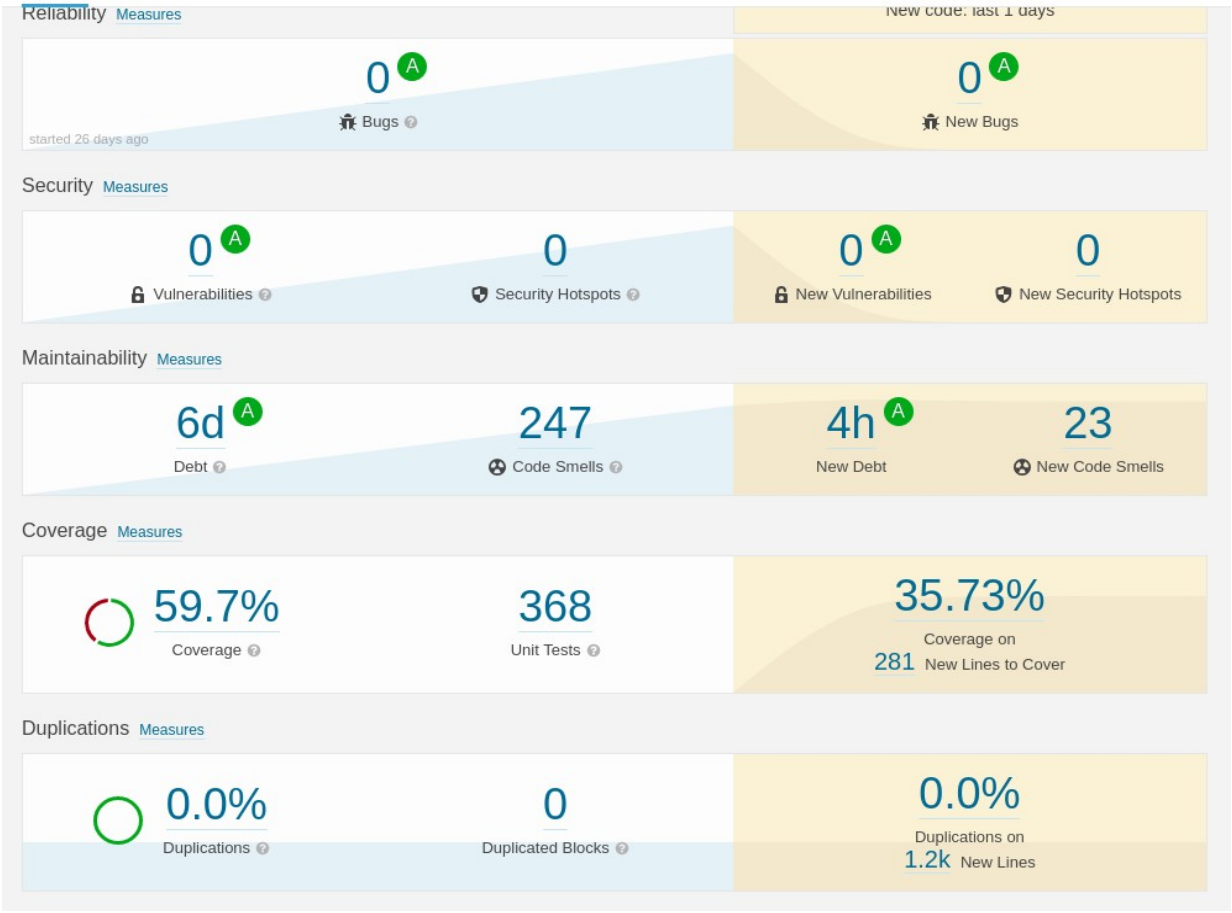


Рис.3.11. Результат статичного аналізу якості коду Sonar

У ході проведення активного рефакторингу під час реалізації програмного забезпечення було досягнуто покриття тестами в майже 60% з загальною кількістю тестів більше 350, що дозволило виявити більше 20 дефектів на етапі модульного і інтеграційного тестування і таким чином не допустити збоїв у роботі всього інтегрованого комплексу DLT рішення, бо трасування і виявлення дефектів в децентралізованих системах — дуже дорогий і трудомісткий процес, оскільки знаходження причини проблеми ускладнене великою кількістю схожих вузлів і невідомо від кого прийшов збій чи який вузол першим піддався відмові, бо до вузлів у оператора може не бути доступу (журналів чи засобів діагностики на льоту).

Також в рамках даної роботи сумарна кількість рядків коду проекту склала близько 10 тис. рядків не враховуючи клієнтський додаток, що є хорошим показником можливості створення компактних децентралізованих систем з обширними функціональними і безпековими можливостями. Процедура розробки і її

динаміка описана графіком додавання нового коду з його сумарною кількістю, що веде Sonar Cloud і що зображено на рис.3.11.

Рис.3.12. Динаміка розробки Sonar

3.4. Неперервна інтеграція

Неперервна інтеграція (CI) — одна із ключових складових досягнення високої якості програмного забезпечення шляхом зменшення кількості збоїв і дефектів і активного моніторингу стану розроблюваної системи (кількість успішно виконаних тестів і покриття коду тестами, наявність так званих code smell і верифікація того, що поточний інкремент успішно проходить процедуру збірки і тестування всього ПП). Тому у ході розробки ПЗ в рамках даної роботи було використано CircleCI — хмарну платформу виконання операцій по неперервному інтегруванню ПЗ, що дозволяє швидко додати в проект хмарні процедури збірки і верифікації з мінімальними зусиллями і максимальним ефектом від візуалізації процесів.

Так пайплайн останньої збірки на CircleCI показано на рис.3.13.

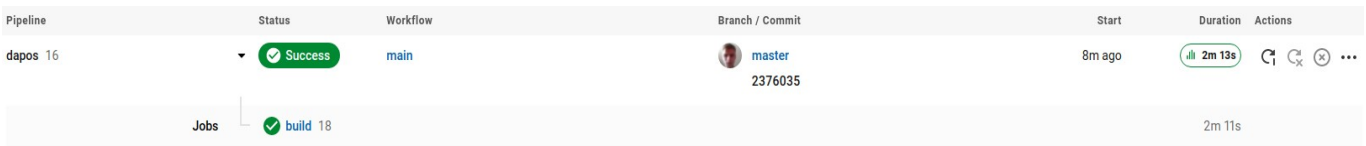


Рис.3.13. Результат успішного виконання пайплайна CircleCI

Пайплайн включає в себе результати роботи однієї задачі по збірці — build, яка в свою чергу об’єднує декілька підзадач по збірці, тестуванню, інтеграції, компонуванню фінальних артефактів і аналізу якості коду засобами Sonar, що описані у попередньому пункті.

Детальний технічний список задач, запущених в рамках пайплайна зображено на рис.3.14.

Сюди входять як суто технічні задачі по конфігурації середовища виконання збірки, завантаження образів docker і коду репозиторія, так і генерація коду GRPC, завантаження залежностей, відновлення кешу попередньої збірки, запуск тестів і аналіз коду на якість засобами Sonar.

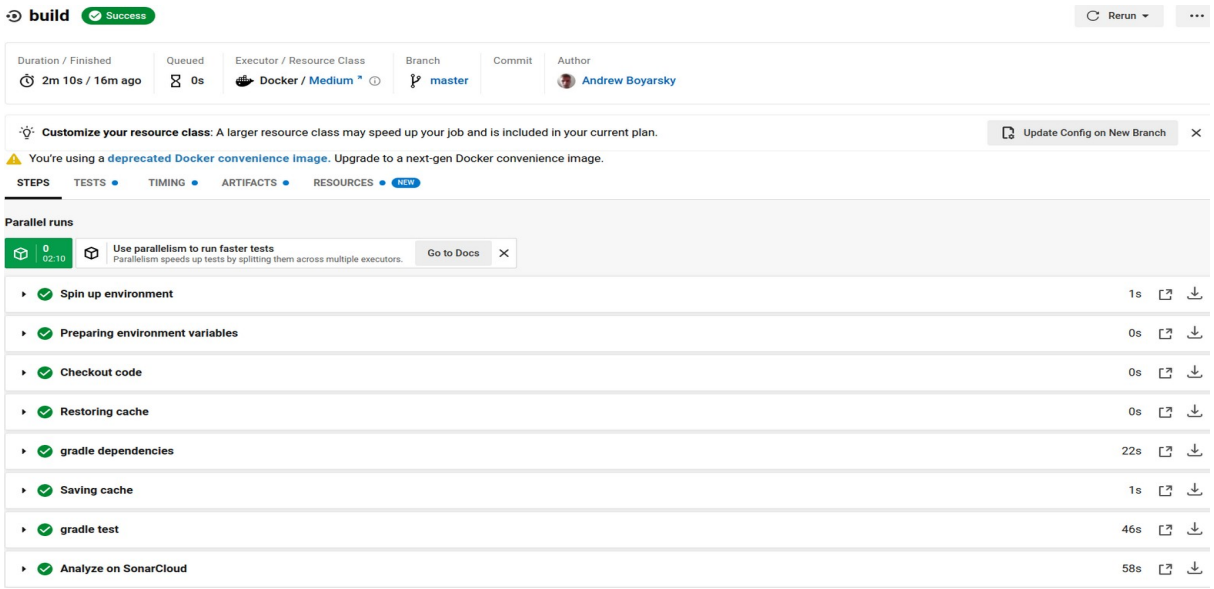


Рис.3.14. Список виконаних задач неперервної інтеграції CircleCI

3.5. Висновки до розділу

У ході реалізації ПЗ системи забезпечення довіри було розроблено один (TS) і удосконалено два окремих програмних продукти Dapos і Adap, кожен з яких реалізований мовою програмування Java відповідно до проекту та специфікації вимог поданих у розділі 2. Було виконано розгортання децентралізованої мережі на базі локального комп’ютера та виконано первинне налаштування конфігурацій для підтримки розгортання і локального тестування. Інтегрована програмна система була покрита модульними та інтеграційними тестами на 60% що дозволило

перевірити її основні бізнес-функції, реалізовані скінченим автоматом та не витрачати час на покриття менш важливих та досяжних більш простим шляхом функцій. Всього розроблено близько 360 модульних тестів, що запускаються в CI/CD пайплайні CircleCI, що дозволяє відслідковувати статуси збірок артефактів, та відразу розгортати їх в хмарі без ручних операцій.

Завдяки застосуванню емпіричних методів програмної інженерії було досліджено статистичні метрики розробленого програмного забезпечення з використанням Sonar і отримано рівень якості — A, що підтверджує високу якість вихідного коду, під час розробки якого було застосовано техніки “чистого коду” і “активного рефакторингу”.

ВИСНОВКИ

У ході виконання магістерської роботи було проведено аналіз існуючих DLT систем, виділено їх сильні сторони та недоліки, приведено систему класифікації для аналізу та порівняння DLT рішень. При аналізі EOS та Hyperledger Fabric було виявлено ряд недоліків: вразливість до форків та фрагментованість мережі, високий ступінь централізації та можливість застосування привілейованих команд для цензури учасників мережу, складність розгортання компактних мереж, відсутність захисту від спам-транзакцій. Визначені недоліки, не дозволяють повноцінно використовувати існуючі DLT рішення у якості системи забезпечення довіри в довільній децентралізованій системі, до якої має доступ довільна кількість учасників. Тому метою даної магістерської роботи була реалізація програмної системи забезпечення довіри для децентралізованих систем з метою покращення продуктивності та захищеності їх кооперації над спільними завданнями для демонстрації роботи якої необхідна також інтеграція в DLT систему Dapos.

Перед безпосередньою розробкою системи забезпечення довіри і її інтеграції в DLT P2P систему була розроблена специфікація вимог до програмного (SRS), що визначила всі характеристики та функції, якими повинно володіти ПЗ після його реалізації. На базі специфікації була побудована гібридна монолітна шестигранна архітектура з клієнт-серверними елементами (P2P, Rest API, WebSocket). Для проектування ПЗ використовувались діаграми UML: варіанти використання, послідовності, компонентів, класів. Всі заходи з проектування викладені в документі SAD.

У ході розробки ПЗ системи було розроблено один і вдосконалено два окремі програмні продукти, що взаємодіють між собою:

1. TS – програмна система забезпечення довіри для довільної децентралізованої мережі, що включає моніторинг вузлів та їх добросовісне виконання завдань, облік виконаних операцій і отриманих нагород, протидія атакам подвійної витрати, повторення операцій і розрахунок доступності вузла

2. Dapos – ПЗ DLT вузла, в який інтегрована система забезпечення довіри TS, що інкапсулює дворівневий консенсус (PBFT + DPOS) P2P мережі, окремий компонент P2P вузла, логіку скінченого автомата консенсусу, Rest API веб-сервер для контролю вузла, моніторингу та відправки транзакцій.

3. Adap – Android додаток, що взаємодіє з Dapos через Rest API і надає графічний інтерфейс користувача для реалізації простого способу відправки транзакцій, перевірки балансу аккаунту, моніторингу мережі і т.і.

У ході роботи було змодельовано механізми роботи шардингу в децентралізованих системах і удосконалено способи динамічного конфігурування і координації мережі шляхом автоматичного налаштування параметрів мережі, необхідних для досягнення затверджених параметрів продуктивності.

Вперше реалізовано сумісність криптографічних еліптичних кривих secp256k1 та ed25519 для декількох типів аккаунтів в рамках DPOS DLT системи для захисту транзакцій цифровим підписом.

Розроблену систему забезпечення довіри в децентралізованих системах можна використовувати в уже існуючих децентралізованих мережах (як це було зроблено в рамках даної роботи) для забезпечення захищеності від атак подвійної витрати, повторення операції, підробки автентичності, ухиляння від виконання взятих на себе зобов'язань; посилення довіри учасників мережі до інших учасників шляхом отримання репутаційних балів та рейтингу сумлінного виконання обов’язків перед іншими учасниками.

В подальшому система може бути удосконалена за пропускною здатністю – паралельна обробка операцій, скорочення часу затвердження операцій, розробка більш продуктивних сховищ стану скінченого автомату. Також може бути покращена в сторону зменшення кількості необхідних надлишкових даних для забезпечення власної стабільності у ході збоїв вузлів (parent-child chains, DAG).

Результати роботи були опубліковані у 2 збірниках тез конференцій: VII Всеукраїнської науково-практичної конференції молодих науковців «Інформаційні Технології – 2020» та 6-ої Східно-Европейської конференції “Математичні та програмні технології Internet of Everything”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2009, [Електронний ресурс]. Режим доступу: https://bitcoin.org/bitcoin.pdf

2. All Cryptocurrencies – Coinmarketcap – [Електронний ресурс]. – Режим доступу: https://coinmarketcap.com/

3. Distributed ledger technology use cases – ITU-T – [Електронний ресурс]. – Режим доступу: https://www.itu.int/en/ITU-T/focusgroups/dlt/Documents/d21.pdf

4. An introduction to Hyperledger – Hyperledger – [Електронний ресурс]. – Режим доступу: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_ Whitepaper_Introduction to Hyperledger.pdf

5. Does Hyperledger Fabric perform at scale? – IBM – [Електронний ресурс]. – Режим доступу: https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale/

6. Corda Enterprise – a next-gen blockchain platform – R3 – [Електронний ресурс]. – Режим доступу: https://medium.com/corda/transactions-per-second-tps-de3fb55d60e3

7. Four reasons why enterprises are looking at the EOSIO blockchain protocol – Block.one – [Електронний ресурс]. – Режим доступу: https://block.one/news/four-reasons-why-developers-and-enterprises-are-looking-at-the-eosio-blockchain-protocol/

8. EOS: An Architectural, Performance, and Economic Analysis – Whiteblock – [Електронний ресурс]. – Режим доступу: https://whiteblock.io/wp-content/ uploads/2019/07/eos-test-report.pdf

9. Class action lawsuit filed against Block.one, the company behind EOS – Brave New Coin – [Електронний ресурс]. – Режим доступу: https://bravenewcoin.com/insights/class-action-lawsuit-filed-against-block-one-the-company-behind-eos

10. Деякі питання оптимізації системи центральних органів виконавчої влади : постанова Кабінету Міністрів України від 2 вересня 2019 р. №829 // Урядовий кур’єр. – від 03.09.2019. – № 167.

11. Про запобігання та протидію легалізації (відмиванню) доходів, одержаних злочинним шляхом, фінансуванню тероризму та фінансуванню розповсюдження зброї масового знищення : закон України від 6 грудня 2019 р. № 361-IX // Голос України. – 2019. – №251.

12. Дія City – Міністерство цифрової трансформації – [Електронний ресурс]. – Режим доступу: https://city.diia.gov.ua/

13. Проєкт Закону України “Про віртуальні активи” – Міністерство та Комітет цифрової трансформації України – [Електронний ресурс]. – Режим доступу: https://thedigital.gov.ua/storage/uploads/files/Проект_Закону України_Про_віртуальні_активи.docx

14. EOS: About EOS Transaction Fees – Exodus Wallet – [Електронний ресурс]. – Режим доступу: https://support.exodus.io/article/1316-eos-about-eos-transaction-fees

15. Instant zero-fee transactions – Pascal coin – [Електронний ресурс]. – Режим доступу: https://www.pascalcoin.org/whitepapers

16. Kannengießer, Niclas and Lins, Sebastian and Dehling, Tobias and Sunyaev, Ali, What Does Not Fit Can Be Made to Fit! Trade-Offs in Distributed Ledger Technology Designs (January 10, 2019). Proceedings of the 52nd Hawaii International Conference on System Sciences. Доступно в SSRN: https://ssrn.com/abstract=3270859

17. Runchao Han, Gary Shapiro, Vincent Gramoli, Xiwei Xu, On the performance of distributed ledgers for Internet of Things, Internet of Things, 2019, 100087, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2019.100087.

18. Antonopoulos A., Demarest R., Futato D., Mastering Bitcoin: Unlocking Digital Crypto-Currencies // O'Reilly – Sebastopol, California, 2015. – 298 с.

19. Bashir I., Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained // Packt Publishing Ltd – Birmingham, UK, 2018. – 656 с.

20. Saleh, Fahad, Blockchain Without Waste: Proof-of-Stake (May 14, 2020). Доступно в SSRN: https://ssrn.com/abstract=3183935

21. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In CRYPTO (Part 1) (LNCS), Vol. 10401. Springer, 357– 388

22. Ващук О., Шувар Р., Плюси та мінуси алгоритму консенсусу Proof of Stake. Відмінності в безпеці мережі у Proof of Work та Proof Of Stake. / О. Ващук, Р. Шувар // Електроніка та інформаційні технології. – 2018. – Випуск 9. – С. 106–112

23. E. Deirmentzoglou, G. Papakyriakopoulos and C. Patsakis, "A Survey on Long-Range Attacks for Proof of Stake Protocols," in IEEE Access, vol. 7, pp. 28712-28725, 2019, doi: 10.1109/ACCESS.2019.2901858.

24. Lo, Yuen and Medda, Francesca, Assets on the Blockchain: An Empirical Study of Tokenomics (January 3, 2019). Доступно в SSRN: https://ssrn.com/abstract=3309686

25. X. Li, X. Wu, X. Pei and Z. Yao, "Tokenization: Open Asset Protocol on Blockchain," 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Kahului, HI, USA, 2019, pp. 204-209, doi: 10.1109/INFOCT.2019.8711021.

26. Nano: A Feeless Distributed Cryptocurrency Network – [Електронний ресурс]. – Режим доступу: http://media.abnnewswire.net/media/cs/whitepaper/rpt/91948-whitepaper.pdf

27. IOTA: Feeless and Free – [Електронний ресурс]. – Режим доступу: https://blockchain.ieee.org/technicalbriefs/january-2019/iota-feeless-and-free

28. The BitShares Blockchain – [Електронний ресурс]. – Режим доступу: https://www.bitshares.foundation/papers/BitSharesBlockchain.pdf

29. Hyperledger Enterprise Solutions: Top 5 Real Use cases – Openledger – [Електронний ресурс]. – Режим доступу: https://openledger.info/insights/hyperledger-enterprise-solutions-top-5-real-use-cases/

ДОДАТКИ

Додаток А

Use case Uml діаграма

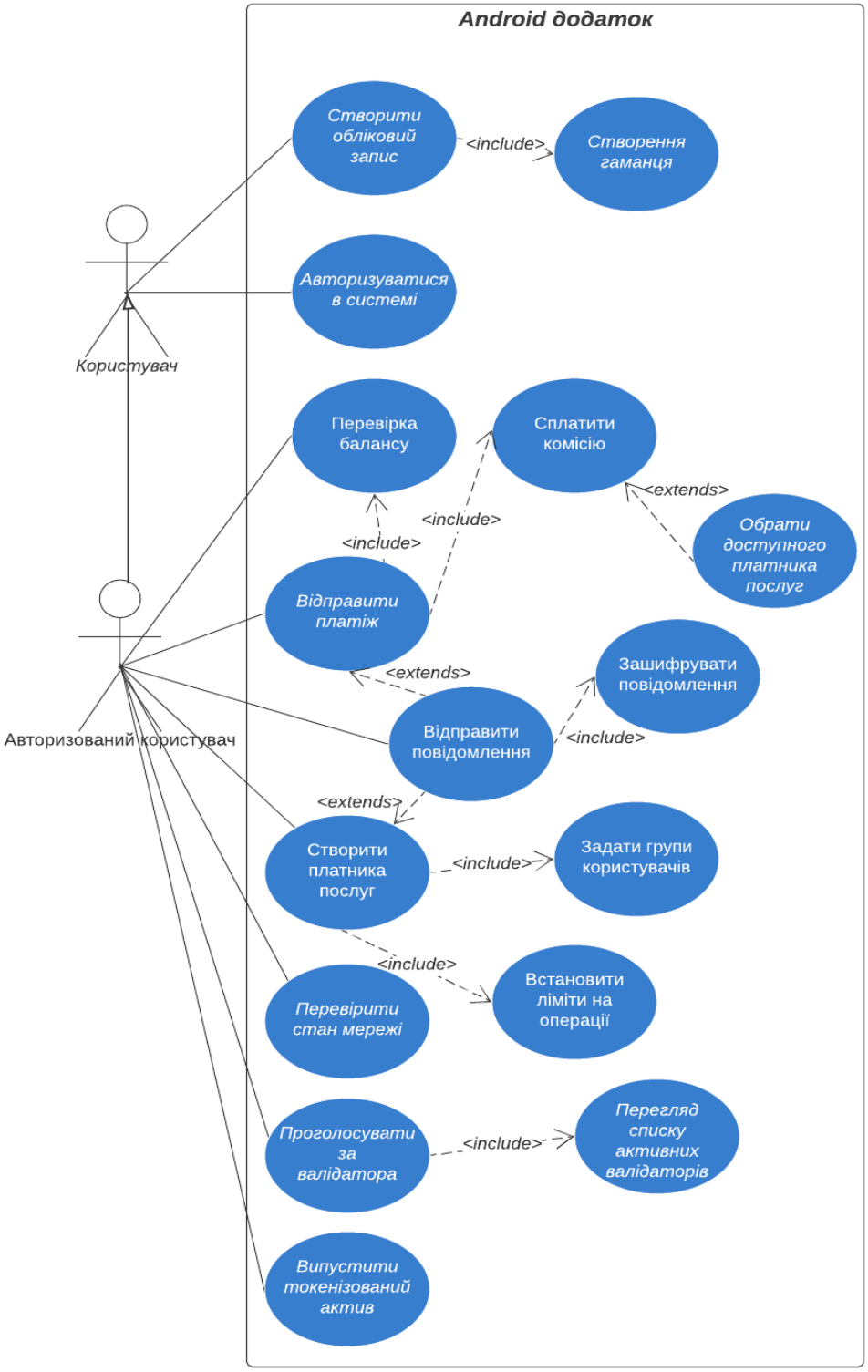


Рис. А.1 UML діаграма варіантів використання Android додатку

Додаток Б

Документ опису архітектури ПЗ

ABC Alphabet LLC

DAPOS

Software Architecture Document (SAD)

CONTENT OWNER: Andrii Boiarskyi

DOCUMENT NUMBER:	RELEASE/ REVISION:	RELEASE/REVISION DATE:
1	v1.0	04-25-2022

Table of Contents

1. INTRODUCTION

This document is intended to describe the architecture of the distributed registry software system with support for parallel computing and asynchronous transactions, which should be an upgrade of the existing dPOS blockchain system with support for micro-payments with zero-commission token assets (DAPOS). The description of the system architecture should include all significant architectural solutions, end-user usage scenarios, components and modules that allow further full software implementation without the need for significant changes to the high-level system architecture.

1.1. Purpose

1.1.1. Problem statement

The field of distributed registry technologies is actively developing and is one of the most promising new branches of science, as it emerged relatively recently, in 2008 with the development of bitcoin as the first completely free of P2P e-cash, since then the number of similar projects (forks) or new implementations of the decentralized register increased to more than 10 thousand. However, the problems with the theoretical performance limitations of any such distributed system lie in the power of single-threaded execution of the main code for receiving and processing transactions by one node, which is actually the bandwidth of the entire network. This factor negatively affects the scalability and applicability of blockchain systems for the tasks of real sectors of the economy.

Instead, the ability to process blocks and / or transactions in parallel will increase the performance of DLT nodes, as it will enable all free node processors to increase bandwidth and speed of tasks sent to the network.

Therefore, within the framework of this document, an architectural description of algorithms for parallel execution of disparate data processing tasks for DLT systems should be created, which will use unused system resources of each node to perform part of the overall work with partial overlap and high level of correctness.

1.1.2. Objectives

The purpose of this document is to describe the architectural basis for the modernization of a decentralized blockchain system to implement distributed tasks with a high degree of parallelism and computational complexity between the DLT nodes of the system.

The upgraded DLT system, in addition to maintaining existing performance and quality indicators, must implement a third-level consensus algorithm (validity of asynchronous tasks) and provide a proven probability of correct execution of each parallel subtask more than 90% with the provision of asynchronous level in case of malicious behavior or failure to perform the amount of tasks for a certain time.

The resulting system should include a modified Android client, REST API to connect this application, DLT node software with subsystems: P2P between nodes, PBFT, dPOS, AJ consensuses, Job Scheduler, Task Checker, high-performance storage subsystem

1.2. Scope

Architectural solutions aimed at developing a DLT system with support for asynchronous transactions and parallel computing should only include the modernization of the existing dPOS blockchain system with developed during the implementation of the algorithm for distribution, execution, compilation and verification of tasks within the DLT network. This upgrade includes the expansion of the existing application for Android.

No additional application programming interfaces should be created (except for the extension of the existing REST API)

Creating a new third-party system, integration with third-party services, the introduction of payment for parallel tasks, as well as the distribution of payments between performers is beyond the designed software

This document should not directly describe the mathematical apparatus of the algorithm of parallelization, distribution, execution, recovery, assembly and completion of parallel tasks, as its technical details are a commercial secret

1.3. Glossary

Term list:

Blockchain: growing list of records, called blocks, that are securely linked together using cryptography. Each block contains authorized state-transitions (user actions) – transactions, which are signed by sender (initiator) and executed withing block (block accepted and validated by set of validators, miners depending on consensus algorithm)

DLT: Distributed Ledger Technology, term is wider than blockchain (includes it) and generalize P2P systems which may function without intermediary by working on self-reproductive copy of all authorized actions – ledger (which is the same for all nodes)

Node: Server on the Internet with public IP which run a blockchain software and connects to the other servers via P2P

Java: A high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

NoSQL: database which provides a mechanism for storage and retrieval of data in means other than tabular relations used in relational databases

Acronym list:

Dapos: existing blockchain system with dPOS consensus for micro-payments with zero-commission.

Adap: existing client android application for Dapos

REST: Representational State Transfer, web API featuring a state-less client-server infrastructure.

API: Application Programming Interface, a protocol used as an interface to allow communication between different components.

JSON: JavaScript Object Notation, a text-based standard for human-readable data exchange.

1.4. Stakeholders

The following stakeholders can be identified for the designed software: site administrator, site software developers and android application, users performing parallel tasks and transferring funds and validators participating in the dPOS consensus.

#	Name	Description	Concerns
1	Admin	Run and maintain Dapos	System resources efficiency, reliability and possible maximum uptime with no crashes
2	Developer	Develop and maintain source code of the Dapos and Adap. Write docs	Ease of new features development, variety of supported libraries and frameworks allowed to use, code complexity and performance parameters
3	Validator	Participate in dPOS consensus by risking own funds to validate and accept blocks in network	Fund-burning risks, node stability, expected revenues, transaction fees, required routine tasks to continue block’s validation
4	User	Send asynchronous or parallel tasks for execution,view results, transfer funds	Task execution estimated time, correct results after task executed pieces compilation, no double spending, irreversible transactions, fast transaction confirmations < 5sec

1.5. Non-functional requirements

Designed system must allow at least 20 parallel task execution, 1000+ TPS and 200 active users per node with 100 validators per node.

Other productivity requirements:

- Dapos deployment time should be less than 10s.
- The connection time to the P2P nodes of the network must be less than 30s.
- The transaction speed must be at least 1000 per second at 20 nodes on a network with 4GB of memory and 4 core processors clocked at more than 4GHz.
- The time to create a new block should not exceed 6 seconds.
- Dapos must support a registry size of up to 4 TB.

- The time to execute a simple read request from the database should be less than 3s, when reading the range with pagination or filtering - no more than 15 seconds, when updating data or adding new ones - no more than 5 seconds.
- Transaction confirmation time should not exceed 12 seconds if the pool of unconfirmed transactions is not completely filled.
- The startup time of Adap on devices with Snapdragon 710 and 4 GB of memory should not exceed 5 seconds.

2. ARCHITECTURE OVERVIEW

Under this section all designing decisions are described to achieve goals and requirements under scope given in paragraph 1.

2.1. Architecture summary

Dapos is developed using the 4 + 1 software architecture design method, as it provides a more complete structured description not only of system function implementation options, but also of source code organization (three-layered architecture), deployment of ready-made artifacts (monolith first principle), and interaction of various processes (both in individual components and within one). All these elements of the architecture are combined with custom usage scenarios, and for each of the stakeholders, the usage scenario is different.

To better understand high-level Dapos architecture the following UML component diagram is given below.

We can list the following core components:

1. P2P Node – Server which listen for P2P connections and responsible for transactions and blocks propagation, receiving consensus messages and cooperate on the blockchain. Also it acts as a P2P client for any amount of other P2P Node’s to build up a P2P network

2. State Machine – the core component of the business logic which define transaction formats, check, parse, verify, execute and store state changes in Database via State Store interfaces

3. Database – specific data storing component which lay beyond DAO interface and may have any available implementation and use any of existing database engines: SQL, NoSQL, File, Memory Cache and other

4. Web Server – is special REST API server which accepts requests to send transaction, check balance or blockchain state, schedule parallel task and coverts it into format acceptable by business logic State Machine and then return to the client JSON response. Also it authorizes clients via Supportive functions module by using KeyStore files which stores private keys for signing transactions.

5. Supportive functions module is an optional component of the Dapos system which implements on server-side different useful features, such as keystore, but node may function without it to be more compact and resource-efficient.

6. Android client – Adap, is a separate application which show GUI to the user on Android device and allow easy access to the all features implemented in Dapos via using of Rest API directed to the Web Server component.

As long as 4+1 architecture view define the core concept of it as ‘Scenarios’ we list an UML Use Case diagram for User and Node actors, which is given below. The diagram show us the core functions of the system which must be implemented during development phase, such as:

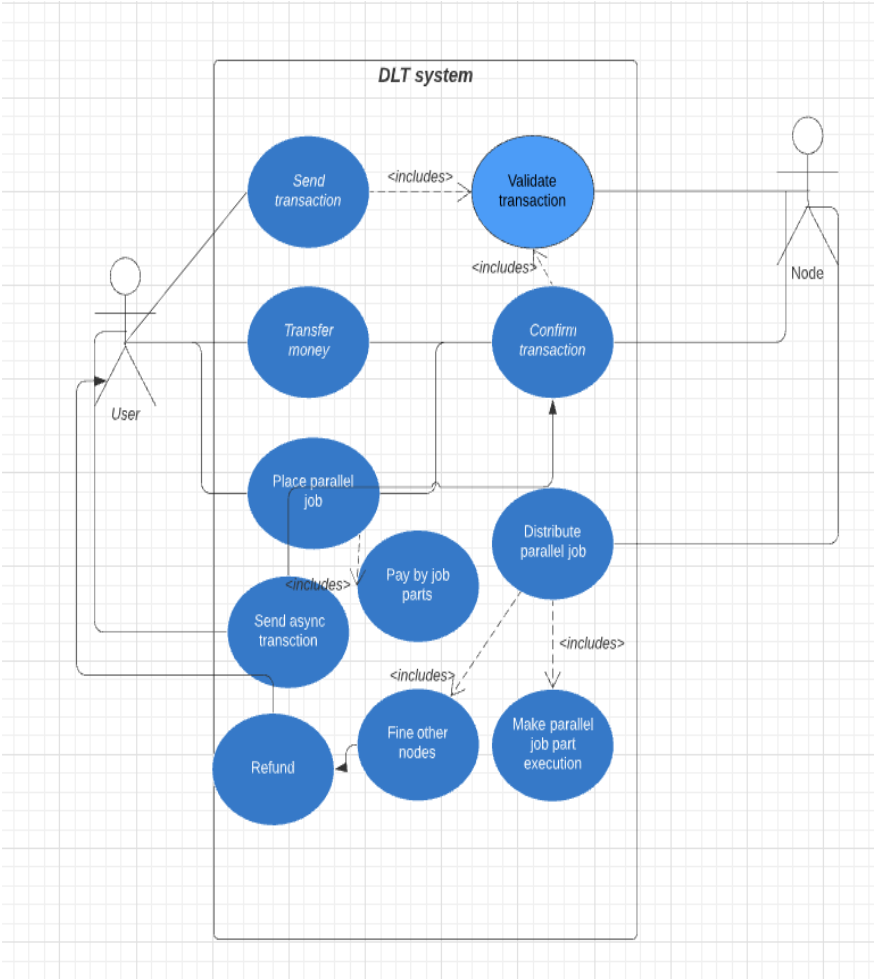
1. Transaction sending feature, transaction must be validated by node and confirmed by P2P network

2. Transfer money feature, which is associated with transaction (the same transaction sending/validation/confirmation routine but with different payload)

3. Place Parallel Job by User, which is also conducted via transaction routine but with more job done by Node, which includes parallel jobs distribution, performing job part (chunk) execution, fining bad nodes which didn’t execute their parts and refunding user indirectly when job for some reason fails (which is possible when network is corrupted or nodes are cheating)

4. Send asynchronous transaction by User almost the same as synchronous transaction but is significantly cheaper and is executed asynchronously in background by the P2P Nodes. It’s not guaranteed that it would successfully finished with 100% probability.

5. Also by polling parallel job state, user pay per job part executed



To describe the process of transaction execution and parallel task workflow a few UML Sequence diagrams are designed.

Transaction Sending

The user initiates the payment by specifying his password, the address of the account from which he wants to send funds, the recipient, the amount, additional message and its encryption parameters (keys to be used), the web server receives the user's request and checks all parameters for validity.

After performing such a check, the web server tries to authorize the user by verifying the correctness of the specified password phrase through the keystore component included in the add-on module, if the user entered the password and account ID correctly, the web server forms the body of the transaction spent gas (abstract value, indicating the complexity of the operation) on such a transaction by calling the component State Machine. After receiving the gas quantity parameter, the web server assigns this transaction value as the maximum possible amount of gas for its execution and sets the gas price taken as the current average for the network. The web server then calls an authorized keystore to sign the transaction on behalf of the user. The signature is executed so that transactions can be performed only by the owner of the private key to the account, and forgery or alteration of the transaction content (including the signature) resulted in the invalidity of the transaction.

The signed transaction is sent to the P2P node, where it enters the pool of unconfirmed transactions, which are selected in a non-deterministic way to confirm when creating a block. All transactions must be re-validated on a finite machine before entering the pool, which is similar to the one performed by the web server before sending it, but such validation may be light-hearted and do not require operations that read the state of the state machine.

Signed transaction after entering the pool - is distributed among other nodes by sending to the nearest n neighbors (the widest channel, the lowest response delay).

As long as the transaction is in the pool, it spreads to other nodes. The web server notifies the web server that the transaction has been transferred to the network. From this point on, the web server starts polling the P2P node in a loop with a certain period (eg 3c) about the status of the transaction (accepted / rejected, confirmed / unconfirmed).

Until the transaction is confirmed (in the case of an unfilled pool of unconfirmed transactions, this time should be up to 2 block generation periods), the web server is waiting for its confirmation.

When a P2P network accepts a new block, the P2P node scans all transactions in that block and runs them on a state machine. blocks. The result of the transaction is transmitted in response to the user.

Parallel Task Execution

The process of sending a task for parallel computing is sent in the form of a normal transaction, which specifies the input data for the task, its type, priority and commission for execution.

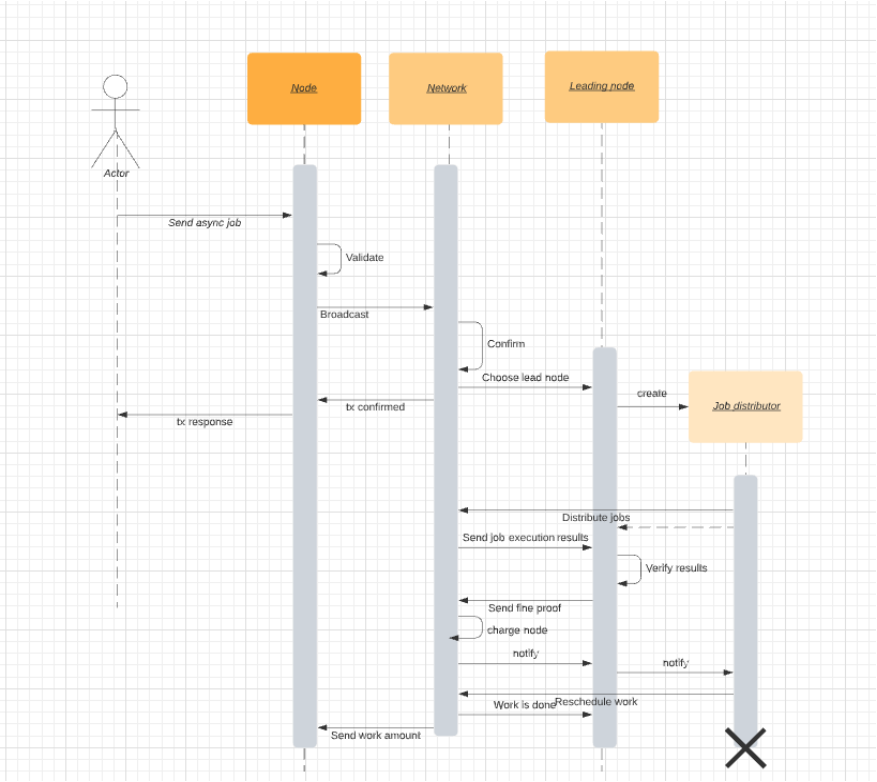
If the DLT nodes have sufficient resources, they take on the task by randomly selecting a leading node that distributes work among other nodes and monitors its performance, in cases where the controlling node fails in its task, any executive node can take its place by providing cryptographic proof of errors of the current leading node with a subsequent penalty in the consensus of the second level for such a leading node.

Every executed and verified part of job by leading node is payed by user from a secured deposit account for this task.

Any failed or executed incorrectly task is rescheduled for execution up to 3 times.

When task fails more than 3 times, then the whole set of chunked tasks is failed and the rest of secured task deposit is refunded to user.

UML-sequence diagram for this process is given below.



2.2. Concerns and decisions

To implement the DLT system described in the previous section, the Java programming language will be used, as this programming language allows you to develop cross-platform applications for different operating systems and computer architectures, and unlike C ++ has a better level of security and automates or delegates routine operations with virtual machine memory, which significantly reduces the number of possible errors and increases the testing and speed of application development.

The main reason for choosing Java is high portability and compatibility, which will achieve a high level of software availability on different platforms, while having only one instance of compiled code.

Another reason for choosing the Java programming language is the extensive ecosystem of third-party projects, frameworks and libraries in the public domain, which will allow you to implement the final software product of any complexity in a relatively short time.

The last reason for using Java is the author's professional experience in developing the Apollo blockchain programming language in this language, which will allow him to focus more on achieving goals rather than exploring the capabilities of libraries and frameworks.

The following software tools were chosen to implement the main tasks:

1. IntelliJ IDEA - integrated software development environment in Java programming language, used for coding, configuration and testing of software;

2. Jetbrains Xodus is a high-performance NoSQL database with a snapshot isolation level, better random read and write speeds than built-in SQL counterparts (H2, Apache Derby, HSQLDB) and support for key-value and entity repositories;

3. Spring Boot - a framework that automates the creation of an application based on the Spring Framework and provides automatic deployment of a servlet web server, dependency inversion container, security and logging configuration, database connection and proxying components to support AOP.

4. Apache Tomcat - a web server to provide the REST API for the ability to send transactions through the client application.

5. Gradle - a tool for building a project, running unit tests, connecting dependencies and generating artifacts for simplified launch and distribution of software;

6. Git - version control system, used to save project data, recover lost code and keep a history of software versions.

7. CircleCI - cloud system of continuous integration, which launches the assembly of the project in the cloud with each change in the state of the repository and allows you to connect third-party services and tasks forming a pipeline.

8. SonarCloud - a cloud system of intelligent analysis of software code, which estimates the number of possible defects, security risks, violations of programming conventions and shows how long it takes to eliminate them.

9. Jacoco - a plugin for building Gradle project, which generates a report on the coverage of code tests (lines, classes, packages, branches).

10. GitHub is a cloud provider of Git repositories, which allows you to connect cloud services to projects and monitor project status (assembly, test coverage, code quality).

3. COMPONENTS

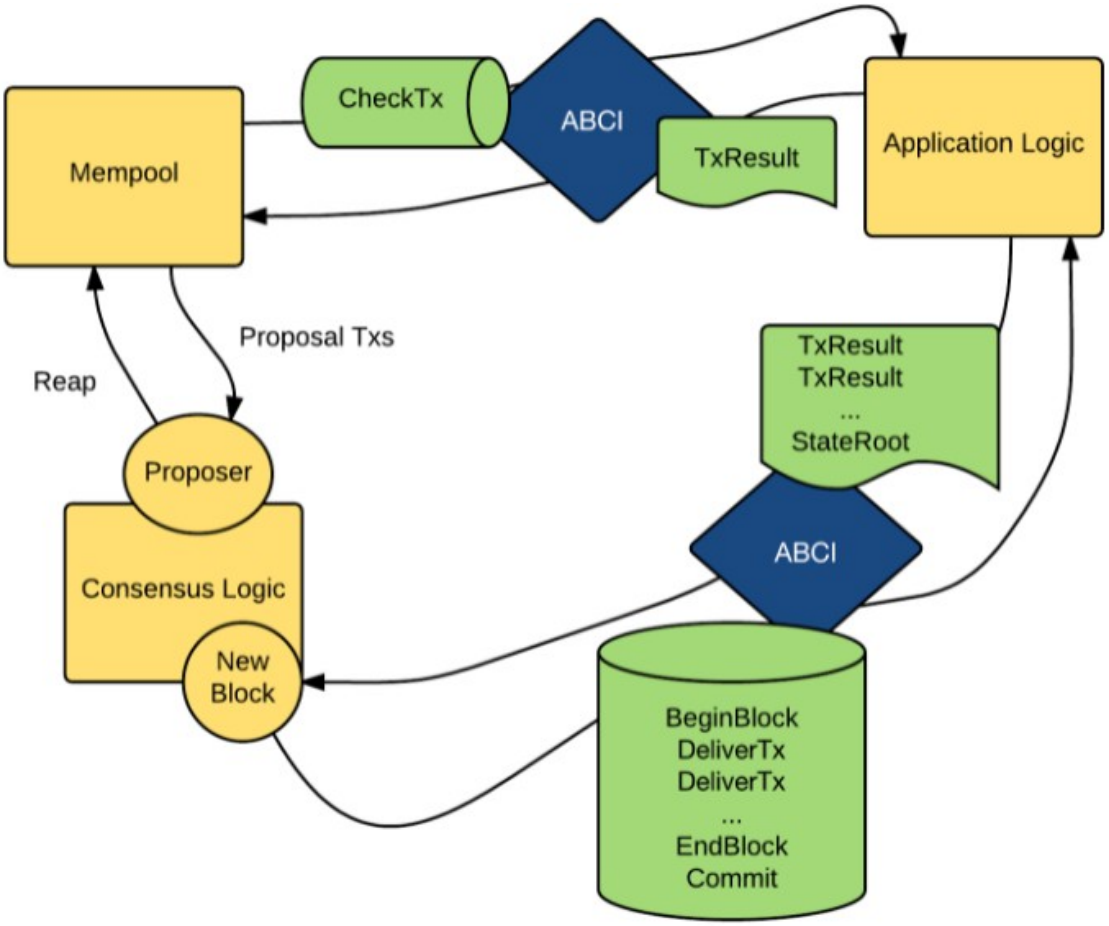
In previous paragraphs we listed 6 core software components which forms a Dapos + Adap applications. In the final lookup we ca group these theoretical concepts into real subsystems and list it as an independent software applications.

Tendermint Node Go application

Consist of P2P communication protocol, server and client, responsible for messages replication across the whole network, responsible for maintaining P2P connections with other Tendermint Node and implements the PBFT round consensus which allow reliable finish state-transitions replication with arbitrary input data.

Tendermint setup a special framework to work with, and define interface: ABCI which must be implemented by client application in order to create finished system.

Consensus round along with tendermint framework is given on the following figure below.



Dapos Service

Consist of implemented ABCI interface of the Tendermint Node, responsible for starting Tendermint Node and applying configurations to it, implements second level consensus dPOS by configuring Tendermint validator with a different validator powers depending on number of owned delegated stake by users

Also it defines state store for user balances, validator powers, chain configuration which may be changed over time and genesis configuration.

Provides REST API and Swagger client to allow easy access to transaction sending feature.

Control the start of the tendermint node and maintain stable connection via GRPC and generated protobuff entities

On the package diagram we can overview the high-level packages and features associated with them: repository, cryptography operations, transaction execution and dispatching, configuration capabilities, GRPC connectivity, genesis and account ledger features.

4. TESTING

Testing of the developed system should be based on a multilevel testing model, so the minimum set of tests should be written by developers - these **are modular tests** that should cover more than 90% of lines and branches of code, modular tests are performed at each project assembly and immediately show new iteration.

The next level of tests is integration, they are developed in two stages - by developers to test localized scripts between classes and testers by automators who test the user usage scenario directly either through the REST API or through the GUI using Selenium.

Integration tests written by developers are launched in CI / CD pipelines after each committee in the main branch of develop, stage, master and for each release. Their status is displayed in the main project window, usually in the README file. These tests are included in the mandatory set of criteria for passing the new function before adding it to a single code base.

End-to-end integration tests run on a separate environment asynchronously when adding new changes to the develop / stage / master branch

Component tests are usually scripts for manual testing that are at the junction of several systems and must be run after each increment by both testers and developers.

Performance tests are a mandatory component and are conducted by testers in separate specially organized environments, tests are launched automatically after adding new changes to the develop branch. The report is published by mail to developers, testers and management.