НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Розрахунково-транзакційна система програм лояльності клієнтів на основі технологій смарт-контрактів

Виконав (-ла): студент (-ка) 4 курсу, групи ІО-83
(шифр групи)

Омелянський Антон Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник професор, д.т.н., проф. Жабін В.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) проф., д.т.н., Симоненко В.П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент декан ФПМ, д.т.н., проф. Дичка І.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Омелянського Антона Сергійовича

1. Тема проєкту *Розрахунково-транзакційна система програм лояльності клієнтів на основі технологій смарт-контрактів*
керівник проєкту професор, д.т.н., проф. Жабін В.І.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____ 2022 року №_____
2. Термін здачі студентом закінченого проєкту *20 травня 2022 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні дані*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Опис предметної області, дослідження методики побудови розрахунково-транзакційних систем на базі технології смарт-контрактів, програма забезпечення клієнтських транзакцій та розрахунків між організаціями, програма мобільного гаманця клієнта*
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, діаграма класів системи, схема алгоритму платіжної транзакції, схема алгоритму розрахунків між організаціями

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2022-25.03.2022*	
4.	*Розробка структур окремих підсистем*	*25.03.2022-05.04.2022*	
5.	*Програмна реалізація системи*	*05.04.2022-15.04.2022*	
6.	*Оформлення пояснювальної записки*	*15.04.2022-20.05.2022*	
7.	*Захист програмного продукту*	*25.04.2022*	
8.	*Передзахист*	*03.06.2022*	
9.	*Захист*	*21.06.2022*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

В бакалаврському дипломному проєкті реалізовано систему, призначену для виконання платіжних транзакцій з функцією обміну балів між різними програмами лояльності.

Програма дозволяє обмінювати різнотипні бали лояльності (бонусні бали, милі, літри тощо) для використання їх споживачем в будь якій торгівельній мережі, підключеній до даної екосистеми. Для забезпечення розрахунків між власниками програм лояльності та встановлення довірчих відносин між ними використані технології смарт-контрактів та криптовалютні рахунки.

Смарт-контракти створені на мові Java в блокчейн IBM Blockchain Platform.

Мобільний гаманець с картками лояльності розроблений для Android пристроїв. Для взаємодії с платіжним терміналом в торговельній мережі пристрій використовує технологію NFC.

Annotation

System performing payment transactions with the function of exchanging points between different loyalty programs was realized in the Bachelor's Degree project.

The program allows you to exchange different types of loyalty points (bonus points, miles, liters, etc.) for use by consumers in any retail network connected to this ecosystem. Smart contract technologies and cryptocurrency accounts were used to ensure settlements between loyalty program owners and to establish trust between them.

Smart contracts were realized in Java language on IBM Blockchain Platform.

Mobile wallet with loyalty cards was designed for Android devices. The device uses NFC technology to interact with payment terminal in retail network.

ОПИС АЛЬБОМУ

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1			Документація загальна		
2			Знову розроблена		
3					
4	А4	ІАЛЦ.467200.002 ТЗ	Розрахунково-транзакційна	3	
5			система програм лояльності		
6			Технічне завдання		
7					
8	А4	ІАЛЦ.467200.003 ПЗ	Розрахунково-транзакційна	77	
9			система програм лояльності		
10			Пояснювальна записка		
11					
12	А1	ІАЛЦ.467200.004 Д1	Розрахунково-транзакційна	1	
13			система програм лояльності		
14			Структурна схема системи		
15					
16	А1	ІАЛЦ.467200.005 Д2	Розрахунково-транзакційна	1	
17			система програм лояльності		
18			Діаграма класів смарт-		
19			контрактів		
20					
21	А1	ІАЛЦ.467200.006 Д3	Розрахунково-транзакційна	1	
22			система програм лояльності		
23			Схема алгоритму платіжної		
24			транзакції		
25					
26	А1	ІАЛЦ.467200.007 Д4	Розрахунково-транзакційна	1	
27			система програм лояльності		
28			Схема алгоритму розрахунку		
29			між організаціями		
30					

Зм.	Арк.	№ докум	Підпис	Дата	ІАЛЦ.467200.001.ОА		
Розробив		Омелянський А.			*Розрахунково-транзакційна система програм лояльності* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Жабін В. І.					
Н. контр.		Симоненко В.П.				НТУУ «КПІ» ФІОТ ІО-83	
Затверд.		Жабін В. І.					

ТЕХНІЧНЕ ЗАВДАННЯ

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Це технічне завдання поширюється на розробку розрахунково-транзакційної системи програм лояльності клієнтів на основі технологій смарт-контрактів. Область використання: торгівельні мережі та інші сфери в яких використовуються бонусні системи лояльності клієнтів. Після конфігурування може використовуватися фінансовими установами в якості транзакційної системи переказу коштів.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту по освітньо-професійної програми “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка розрахунково-транзакційної системи призначеної для обміну балів між програмами лояльності та проведення взаємних розрахунків між організаціями - власниками цих програм.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література та публікації Інтернеті по транзакційним системам, розподілених мережах блокчейн і смарт-контрактах.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Підтримка різнотипних бонусних програм лояльності (бали, милі, тощо).
- Відсутність можливості втручання адміністратором системи в правила обміну балів між програмами лояльності.
- Забезпечення прозорості розрахунків між учасниками без можливості несанкціонованого втручання в процес розрахунку чи модифікації транзакції.
- Заборона доступу до критичної інформації програми лояльності (бази клієнтів та їх активності) для інших учасників системи.
- Можливість використання апаратної частини учасників системи для здійснення ними додаткового контролю за розрахунками та транзакціями.

- Виконання не менше 100000 транзакцій за добу.
- Час непрацездатності системи на протязі року – не більше 72 годин.

5.2. Вимоги до програмного забезпечення

- Вузол блокчейн мережі: розмежування доступу до даних, доступ тільки авторизованим користувачам, використання PKI та X.509 сертифікатів, підтримка смартконтрактів.
- Сервер керування гаманцями клієнтів: взаємодія з іншими модулями згідно специфікації RESTfull, підтримка OAuth 2.0, підтримка докеризація та кластеризації вузлів серверу.
- Мобільний гаманець клієнта: ОС Android 8.0 та вище, взаємодія с платіжним терміналом згідно специфікацій ISO/IEC 14443 та ISO/IEC 7816.

5.3. Вимоги до апаратної частини

- Вузол блокчейн мережі: ЦП 2.4 ГГц, ОЗП 4 Гб, дискова пам’ять 30 Гб, підключення до мережі.
- Сервер керування гаманцями клієнтів: ЦП 2.4 ГГц, ОЗП 2 Гб, дискова пам’ять 10 Гб, підключення до Інтернет.
- Мобільний гаманець клієнта: пристрій Android, NFC, підключення до Інтернет

6. ЕТАПИ РОЗРОБКИ

	Дата
6.1. Вивчення літератури	23.12.2021
6.2. Складання і узгодження технічного завдання	14.01.2022
6.3. Аналіз структури розрахунково-транзакційної системи	28.02.2022
6.4. Створення модулів розрахунково-транзакційної системи	09.04.2022
6.5. Тестування модулів розробленої системи	18.04.2022
6.6. Відлагодження та виправлення помилок	02.05.2022
6.7. Оформлення документації дипломного проєкту	20.05.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЛ – програма лояльності

LS – Loyalty System, система програми лояльності

WMS – Wallet Management System, модуль керування гаманцем клієнта

TSP – Token Service Provider

JWT – JSON Web Token

API – Application Programming Interface

DLT – Distributed Ledger Technology

PoW – Proof-of-Work

PoS – Proof-of-State

BFT - Byzantine Fault Tolerance

JSON – JavaScript Object Notation, текстовий формат даних

REST - Representational State Transfer, стиль взаємодії компонентів розподіленої програми

ЦСК – центр сертифікації ключів

ВСТУП

В сучасних умовах бізнес стикається з жорсткою конкуренцією за ринок. Споживач має можливість вибирати з безлічі однотипних послуг та продуктів з приблизно однаковою якістю та ціною. Продавати своїм наявним клієнтам набагато простіше ніж новим. До того ж утримання наявного користувача коштує в 3-10 разів дешевше, ніж залучення нового. Наприклад, за даними WordStream [1], малий бізнес витрачає тільки на платну рекламу Google для залучення нових клієнтів від $1000 до $10000. За даними Forbes [2], 79% споживачів звертають увагу на наявність додаткових стимулів при обслуговуванні та можуть перейти до конкурента при їх відсутності. Для втримання споживачів в основному використовуються програми лояльності – системи винагородження та заохочення існуючих клієнтів. Дані програми дозволяють зробити випадкового покупця постійним, збільшити середню суму чека та частоту покупок.

Однак впровадження та обслуговування системи лояльності може бути витратним заходом особливо для малого бізнесу. Великих зусиль (в тому числі рекламних коштів) потребується для розширення клієнтської бази. Крім цього, з різних причин (переїзд клієнта в інше місце, відсутність потреби в даному виду товару) споживач не повністю використовує свої бонуси лояльності. А як показано в дослідженні Deloitte [3], навіть тільки сама можливість використати бали в іншому місці, або для інших потреб суттєво підвищують лояльність клієнтів. До того ж спрацьовує «синергетичний» ефект і розширення бази клієнтів з інших мереж перекриває втрати від використання бонусів в інших програмах. Компанії, які це зрозуміли почали перехід від індивідуальних до коаліційних програм лояльності. Однак використання в коаліційних програмах посередників веде до додаткових витрат (іноді доволі суттєвих) та виникненню проблеми довірчих відносин. Крім того виникають труднощі з конвертацією балів з однієї програми лояльності в іншу. Карткові платіжні системи в співпраці з фінансовими установами пропонують використання для цих цілей механізму «кешбек» з зарахуван-

ням фіатних коштів (гривень, доларів, тощо) на банківські картки клієнтів при покупках. Однак цей варіант не працює при готівкових розрахунках і до того ж не дає можливості торгівельним мережам розширювати базу клієнтів лояльності (фінансові та платіжні системи не дають доступу до бази своїх клієнтів). В зв'язку з приведеними вище труднощами на даний час впровадження коаліційних програм лояльності суттєво обмежене і здебільшого включає в себе до 10 учасників, до того ж як зазвичай які мають спільного власника, або не є конкурентами.

З появою P2P мереж блокчейн та технології смарт-контрактів тема створення екосистем з сотнями та тисячами програм лояльності знову стала актуальною [3]. Використання ними математичного апарату розподіленого реєстру DLT (Distributed Ledger Technology) та комп'ютерного алгоритму автоматичного виконання контракту дозволяє виключити посередника між програмами лояльності та забезпечити довірчі між учасниками системи. В даних системах виключається можливість внесення змін в реєстр транзакцій або модифікація чи відмова в виконанні програми смарт-контракту. До того ж кожен з учасників має можливість мати на своїй технологічній площадці реєстр транзакцій, яким можна скористатися при виникненні спірної ситуації. Прозорість правил роботи системи (публічний доступ до алгоритмів смарт-контрактів) забезпечує швидке та просте підключення нових учасників без необхідності додаткових регламентуючих інструкцій. До того ж мережа блокчейн на прикладі роботи з криптовалютами довела спроможність в online режимі обробляти велику кількість транзакцій з практично необмеженою масштабованістю.

РОЗДІЛ 1. ПОБУДОВА КОАЛІЦІЙНИХ СИСТЕМ ЛОЯЛЬНОСТІ КЛІЄНТІВ

1.1. Програми лояльності клієнтів

Програма лояльності клієнтів (ПЛ) представляє собою бізнес-процес взаємовигідних відносин між споживачами і компанією. Її стратегічною метою є збільшення прибутковості сегменту постійних покупців та утримання клієнтської бази. Головними складовими програм лояльності є:

- ідентифікація клієнта (клієнтська база даних);
- стимулювання необхідної поведінки споживача (механізм привілеїв);
- утримання клієнта (механізм комунікацій з клієнтами);
- аналітичний аналіз впливу поведінки клієнта на бізнес та прогнозування розвитку подальших відносин.

Більшість програм лояльності носить ціновий характер, так як вони легше піддаються процесу автоматизації та дають швидкий ефект від використання. Види цінових програм лояльності представлені на рис.1.1.

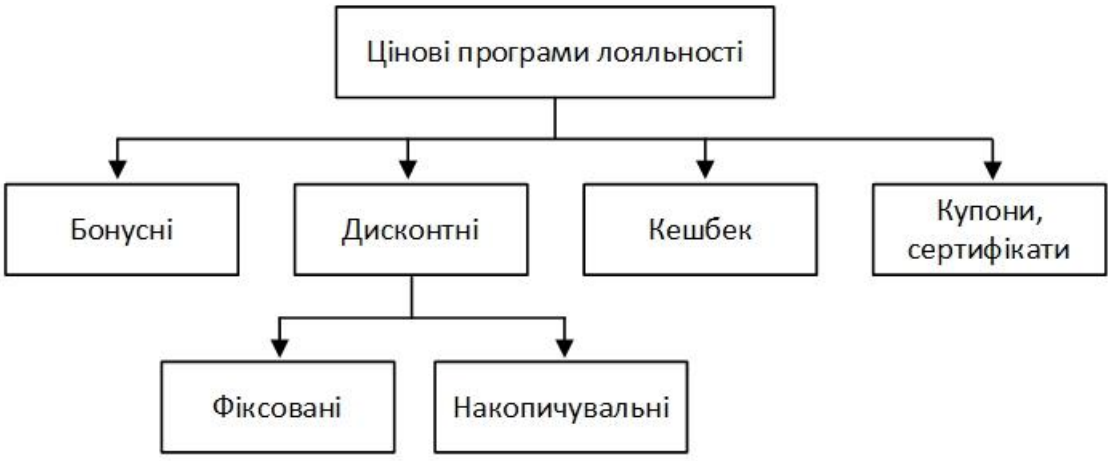


Рисунок 1.1 – Види цінових програм лояльності

Фіксована дисконтна програма – надає споживачу знижку у вигляді відсотка від вартості товару, причому величина відсотка не змінюється в залежності від частоти придбання або суми попередніх покупок клієнта.

Накопичувальна дисконтна програма – також надає знижку у вигляді відсотка від вартості товару, але величина його залежить як часто і на які суми здійснювалися попередні покупки споживачем.

Бонусна програма – після здійснення покупки споживачу нараховуються на його рахунок деяка кількість умовних балів (бонуси, очки, милі, літри тощо). В подальшому дані бали можуть бути використані для отримання безкоштовно іншого товару, придбання його зі знижкою на відповідну суму, отримання додаткових послуг.

Програма кешбек – використовується фінансовими установами для стимулювання здійснення покупок платіжними картками. На відміну від дисконтних програм повернення клієнту частини вартості товару здійснює не продавець, а емітент платіжної картки (за рахунок комісії, яку банк отримує за здійснення транзакції банківським платіжним терміналом).

Купони і сертифікати – одноразова можливість отримати знижку на товари (купони), або придбати товарів на вказану в сертифікаті суму. Зазвичай використовуються як додаткові програми лояльності в період проведення рекламних акцій.

Комплексні програми – об’єднання декількох програм лояльності в рамках клубу постійних покупців (єдиної клієнтської бази). Прикладом її являється бонусна програма для пасажирів аеропорту Хітроу Heathrow Rewards. Її учасники отримують бали за здійснення покупок, відвідування ресторанів, обміну валюти, сплати парковки та можуть їх використати для отримання знижок на сервісу аеропорту, або обміняти на милі авіакомпаній-учасників програми.

Здебільшого для користування клієнтами програмою лояльності компанія випускає для них картки лояльності, але в останній час все більшою популярністю користуються мобільні додатки з підтримкою програм лояльності. Наприклад, в Україні: «Власний рахунок» (торгівельна мережа Сільпо), «Фора клуб» (мережа магазинів Fozzy Group), «Фішка» (АЗС «ОККО», мережа магазинів «Алло», аптеки «Доброго дня») та інші.

Впровадження індивідуальних програм лояльності створених компанією всередині себе і тільки для своїх клієнтів потребує значних зусиль і коштів (обладнання, персонал, закупка програмного забезпечення тощо). Тому компанію почали об'єднуватися в групи, в яких знижки і бонуси можна одержувати у різних учасників такої групи.

В коаліційних програмах з'являється третя сторона – координатор системи, який організує роботу учасників системи та забезпечує діяльність системи в цілому. За рахунок коаліційного ефекту зазвичай зменшуються витрати на підтримку програми лояльності всередині компанії, збільшується притік нових споживачів (обмін клієнтськими базами з партнерами) та створюються додаткові можливості реалізації бонусних програм. Однак в компанії виникають додаткові витрати, пов'язані з необхідністю оплати координатору його операційної діяльності, інтеграції існуючої системи з іншими партнерськими програмами. Також великого значення потребує і забезпечення довірчих відносин між учасниками і особливо до координатора коаліційної програми. Форми коаліційних програм лояльності представлені на рис.1.2.

Рисунок 1.2 – Форми коаліційних програм лояльності

Незалежний оператор створює та координує роботу програми лояльності без прив'язки її до конкретного бізнесу.

Партнерські програми формуються з урахуванням стиля життя клієнта. Якщо програма розробляється для конкретної компанії, якій потребуються додаткові сервіси для більш повного задоволення клієнта, то вона виконує

роль якоря, навколо якого групуються інші (бутики, ресторани, кінотеатри, автосервіси, парковки). Аналогічним чином може бути сформований пул сервісних компаній в рамках корпорації або бізнес об'єкту. Ко-брендингові програми лояльності випускають спільні подарункові або карти лояльності, які приймаються усіма компаніями учасниками проекту. Прикладом такої програми являється випуск банківської картки, яка надає знижку в партнерській торгівельній мережі («картка АТБ від Райфу» - спільний проєкт Raiffeisen Bank Aval і магазинів АТБ).

Однією з нагальних проблем коаліційних програм лояльності є обмін бонусних балів між програмами різних учасників системі. Доводиться враховувати різні одиниці виміру (бали, літри, милі) та різні ступені точності (цілі, до сотих, до тисячних тощо). Крім того необхідно встановлювати курси обміну між усіма бонусними програмами. Це призводить до того, що в реальних системах, або адміністративно вводять єдиний тип бонусної одиниці, або обмежують їх кількість (зазвичай до десятка). Для ко-брендингових програм та програм з якірною компанією ці обмеження не являються суттєвими. Але в випадку побудови національних та міжнародних екосистем лояльності з необхідністю інтеграції великої кількості існуючих програм вирішення цієї проблеми виходить на перше місце поряд с проблемою забезпечення довірчих відносин. Тому такий великий інтерес представляє можливість використання технології смарт-контрактів та криптовалют для побудови систем коаліційних програм лояльності з необмеженою кількістю учасників та типів бонусних програм.

1.2. Платформи блокчейн

Одним з шляхів вирішення проблеми довірчих відносин між учасниками розподіленої системи являється використання блокчейн мережі. Основою блокчейна слугує розподілений реєстр DLT (Distributed Ledger Technology), який містить колекцію взаємопов'язаних блоків транзакційних даних (ланцюг) разом з їх хешем [4]. Блок – це базова одиниця в блокчейн,

яка складається з заголовку блоку та даних блоку. Заголовок блоку зазвичай містить інформацію про поточний хеш блоку (block hash), хеш кореневого блоку (tree root hash), мітку часу (timestamp), одноразово використовуємо число (nonce) та хеш попереднього блоку (previous block hash). Дані блоку містять деталі транзакцій (адреси відправника та отримувача, сума переказу, комісія, тощо). Для хешування зазвичай використовуються алгоритми хешування SHA-256, SHA-512, Keccak256, Scrypt. Ланцюг блоків представляється деревом Меркла або ациклічним орієнтованим графом. Довіра між учасниками системи базується на неможливості зміни даних блоку блокчейн [5]. По перше, оскільки блокчейн розподілений – зміни повинні бути прийняті усіма вузлами мережі. По друге, повинні бути змінені усі наступні блоки ланцюга, оскільки їх хеші включають в себе хеш попереднього блоку.

Процедура прийняття рішення додавання блоку в ланцюг (зміна стану блокчейну) має назву «протокол консенсусу». Найбільш поширеними є наступні протоколи консенсусу:

1. Proof-of-Work (PoW). Для додавання блоку учасники виконують роботу знаходження хешу, який відповідає певним правилам. Перший хто знаходить відповідну комбінацію отримує можливість додати блок до ланцюга. Зазвичай використовуються в мережах, де учасники абсолютно не довіряють один одному (Bitcoin, Litecoin).
2. Proof-of-Stake (PoS). Валідатори ставлять свої криптоактиви в заставу для можливості підписати блок. Чим більша частка, тим більша ймовірність дозволу створення. Потребує використання криптовалюти.
3. Byzantine Fault Tolerance (BFT). Попередньо обрані валідатори підтримують консенсус (підписують блок) навіть якщо третина їх не працює, або є зловмисниками. Зазвичай використовується в приватних блокчейнах з авторизованим доступом (IBM Blockchain Platform, NEO, Ripple).

Платформи побудовані на блокчейн розрізняються в залежності від типу доступу до даних, методів авторизації учасників мережі та можливості підтримки стану (рис.1.3).

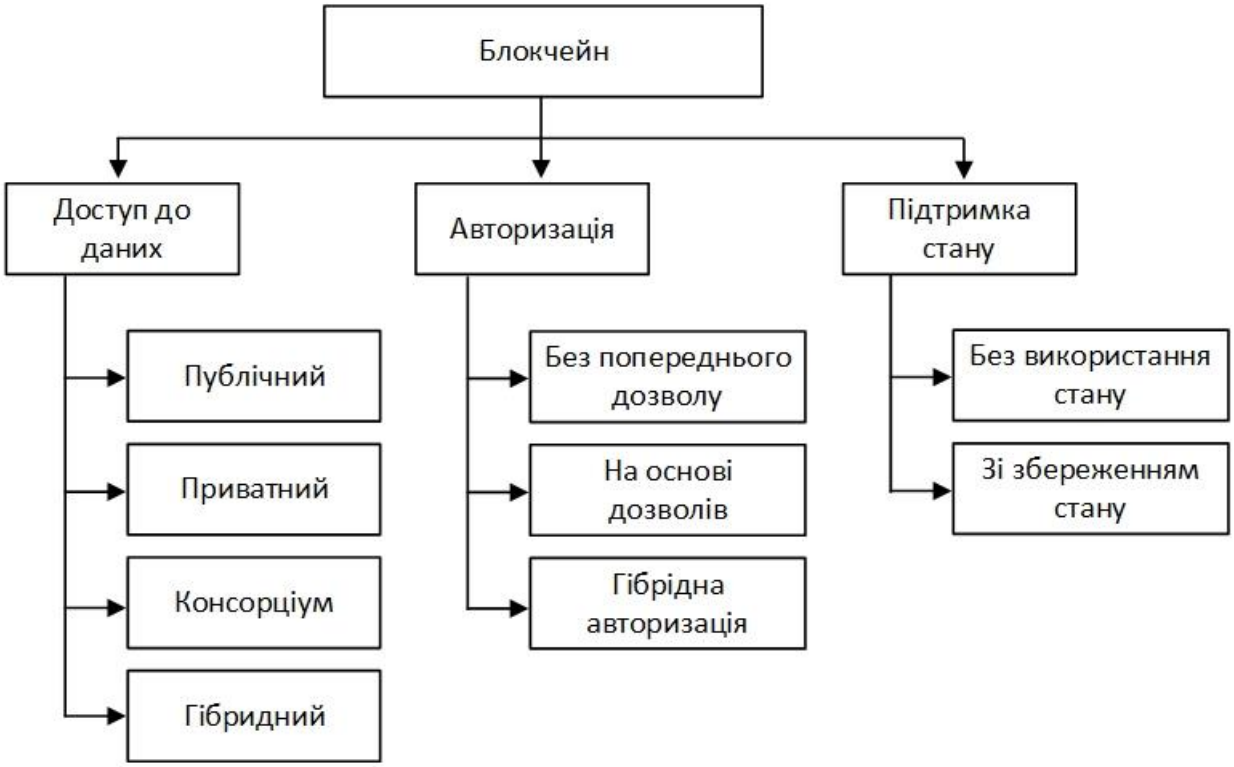


Рисунок 1.3 – Типи блокчейн платформ

В залежності від характеру доступу до даних блокчейн платформи розподіляються на наступні категорії [6]:

- публічний блокчейн (public) – будь-який учасник може отримати інформацію про транзакцію, створити нову та підтвердити транзакцію іншого учасника;
- приватний блокчейн (private) – тільки одна організація та її підрозділи мають доступ до транзакції та її підтвердження в межах однієї групи;
- консорціум (consortium) – кілька організацій утворюють спільноту в межах якої мають право доступу до транзакцій та їх підтвердження;

- гібридний блокчейн (hybrid) – платформа може налаштовуватися для роботи в багаторежимному варіанті (публічний, приватний або консорціум).

В залежності від необхідності авторизації учасників блокчейн платформи розподіляються на наступні категорії [7]:

- без попереднього дозволу (permissionless blockchain) – кожен має можливість приєднання до мережі зі своїми вузлами обчислення;
- на основі дозволів (permissioned blockchain) – для доступу до мережі потрібен попередній дозвіл (авторизація);
- гібридна авторизація – можлива робота без дозволів, але для виконання окремих функцій потребується авторизація.

В залежності від можливості підтримки смарт-контрактів блокчейн платформи розподіляються на наступні категорії [8]:

- без використання стану (stateless) – функціонал зосереджений тільки на перевірці хешів ланцюга блоків;
- зі збереженням стану (stateful) – забезпечує виконання смарт-контрактів та збереження стану їх даних в транзакційних блоках блокчейну.

Найбільш поширені блокчейн платформи приведені у таблиці 1.1

Таблиця 1.1 - Сучасні блокчейн платформи

Назва	Доступ до даних	Авторизація	Підтримка стану
Bitcoin	public	permissionless	stateless
Ethereum	public	permissionless	stateful
Hyperledger (IBM Blockchain)	consortium	permissioned	stateful
Corda R3	consortium	permissioned	stateful
Quorum	consortium	permissioned	stateful
IOTA	public	permissionless	stateless
Ripple	consortium	permissioned	stateless
Sawlooth	hybrid	hybrid	statefull

Технічні характеристики блокчейн платформ приведені у таблиці 1.2.

Таблиця 1.2 – Технічні характеристики блокчейн платформ

Назва	Хеш алгоритм	Ланцюг блоків	СУБД	Консенсус
Bitcoin	SHA-256	Merkle Tree	LevelDB	PoW
Ethereum	Keccak256	Trie	LevelDB	PoS
Hyperledger (IBM Blockchain)	SHA3	Merkle Tree	CouchDB	BFT, PoW, PoS
Corda R3	SHA-256	Merkle Tree	H2	BFT, PoW, PoS
Quorum	Keccak256	Trie	LevelDB	PoS, BFT
IOTA	Winternitz	ацикл.оріент. граф	Trytes	PoW
Ripple	SHA2-512	Merkle Tree	RocksDB	BFT
Sawlooth	SHA-512	Merkle Tree	BlockStore	PoW, BFT

1.3. Технологія смарт-контрактів

Смарт-контракти – це бізнес угоди, описані алгоритмічною мовою програмування, які вбудовані базу транзакцій блокчейн і виконуються автоматично з транзакціями [9]. Смарт-контракти гарантують одній стороні, що інша задовольнить її вимоги згідно правилам прописаним в контракті. Оскільки смарт-контракти можуть мати в собі потенційно небезпечний код для виконання блокчейн платформи, або вимагають для їх створення використовувати спеціалізовані мови програмування з обмеженою функціональністю (мова Solidity блокчейн Ethereum), або виконують їх в окремих контейнерах віртуальних машин (блокчейни Hyperledger, Corda). Для доступу до даних зовнішніх систем в більшості платформ використовується механізм виклику зовнішніх API (оракулів). Єдиний виняток складає платформа Hyperledger, яка не накладає обмежень на доступ до зовнішніх систем (оскільки вона здебільшого використовується в корпоративних системах з наперед відомим складом учасників).

Смарт-контракти оперують з даними транзакції в рамках якої вони виконуються. Додатково деякі платформи (Hyperledger, Corda, Sawlooth) дозволяють в рамках контракту оперувати з приватними колекціями даних. На відміну від даних транзакції приватні колекції даних оперують не світовим станом блокчейну (world state DLT), а журналами транзакцій які перено-

сяться в світовий стан тільки після підтвердження блоку транзакцій. До того ж колекції оперують з комплексними ключами і структурними об’єктами JSON, що дозволяє взаємодіяти з часткою об’єкту колекції та виконувати пошук по значенням окремих атрибутів.

На сьогоднішній день існує достатньо велика кількість успішних проектів використання смарт-контрактів в комплексних програмах лояльності. Так, концерн Lufthansa AG використовує B2B платформу Winding Treeб побудовану на блокчейн Ethereum як інтеграційну платформу розрахунку за готелі, авіабілети та круізи, токенами лояльності. Компанія Singapore Airlines спільно з Korean Air використовують гаманець KrisFlyer, розроблений на базі блокчейну Hyperledger, для обміну бонусними милями.

Висновок до розділу 1

Одним з варіантів вирішення проблеми довірчих відносин між учасниками коаліційної ПЛ є використання платформи блокчейн та технології смарт-контрактів. Оскільки учасники коаліції заздалегідь відомі, зазвичай використовують для розробки такої системи блокчейн с приватним доступом або консорціум. До того ж це дозволяє обмежити доступ до приватної інформації клієнтів ПЛ. З метою забезпечення високої швидкості обробки транзакцій доцільно вибирати платформу з BFT консенсус алгоритмом. До того ж це дозволяє уникнути необхідності використання криптовалюти для підпису блоків DLT. Виходячи з цього в якості платформи для розробки розрахунково-транзакційної системи програм лояльності найбільш придатними є IBM Blockchain Platform (Hyperledger), Corda і Quorum. Ми зупинили свій вибір на IBM Blockchain Platform, оскільки вона дозволяє використовувати для розробки смарт-контрактів універсальні мови програмування Java, JavaScript, має варіанти використання як standalone так і розміщення в хмарі (PaaS) та надає додаткові інструменти обмеження доступу до даних учасників системи (механізм каналів).

РОЗДІЛ 2. ОСНОВНІ ЕЛЕМЕНТИ РОЗРАХУНКОВО-ТРАНЗАКЦІЙНОЇ СИСТЕМИ

2.1. Архітектура розрахунково-транзакційної системи

В даній роботі була розроблена розрахунково-транзакційна система для коаліційних програм лояльності. Для забезпечення довіри між учасниками, а також до організатора системи було прийнято рішення використання блокчейн мережі з виконанням бізнес функціональності за допомогою смарт-контрактів. В зв'язку з необхідністю роботи з конфіденціальною інформацією (клієнтськими базами програм лояльності) вибір було зроблено на корпоративній версії блокчейн мережі IBM Blockchain Platform. Механізм каналів даної платформи дозволив обмежити доступ до клієнтської інформації тільки організатору і власнику програми лояльності. До того ж, використання дозвільного механізму забезпечення консенсусу (permissioned blockchain) дозволило відмовитися від необхідності «майнінгу» криптовалюти, який необхідний в публічних (permissionless blockchain) мережах з механізмами Proof of Work (PoW). Відмова від публічніх мереж та анонімності їх учасників до того ж дозволяє виконувати вимоги законодавства по роботі з персональною та фінансовою інформацією та дотримуватися правил «знай свого клієнта» (KYC) та «протидії відмиванню коштів» (AML).

Розроблена система складається з наступних частин:

- Мобільний гаманець клієнта – мобільний додаток, за допомогою якого користувачі беруть участь в програмах лояльності.
- Модуль керування гаманцями клієнтів – забезпечує об'єднання карток користувача в єдиний обліковий запис «гаманця».
- Обліковий модуль програми лояльності – адаптер до програми лояльності, який дозволяє оперувати з картками лояльності та бонусними коштами клієнтів.
- Транзакційний модуль програми лояльності – смарт-контракт для виконання клієнтських транзакцій по обміну бонусів користувача на криптовалюту.

- Модуль керування програмами лояльності – смарт-контракт для ведення обліку учасників системи (програм лояльності) та керування їхніми рахунками та курсами продажу/покупки бонусів користувачів.
- Модуль операцій в криптовалюті – смарт-контракт для виконання транзакцій з криптовалютою (емісія, покупка, продаж, розрахунки між програмами лояльності).
- Модуль розрахунків між організаціями – смарт-контракт для виконання клірингових розрахунків між учасниками системи.
- Модуль забезпечення консенсусу – підтвердження та фіксація в облікових журналах транзакцій смарт-контрактів.

Організатор системи виконує функції розгортання та адміністрування системи. Завдяки використанню блокчейн та технології смарт-контрактів у нього відсутня можливість модифікації транзакцій, або втручання в виконання бізнес методів контрактів. Але в його задачі входить акцептування деяких дій інших учасників (підключення до системи, зміна ними алгоритму контракту, емісія криптовалюти). Також вузли організатора обов'язково приймають участь у виконанні усіх транзакцій та являються учасниками всіх каналів даних.

Власник програми лояльності для реєстрації в системі звертається до організатора. Для програми реєструється унікальний ідентифікатор програми, а її власнику для доступу в систему видається X.509 сертифікат з приватним ключем. За допомогою даного ключа програма лояльності реєструється в модулі керування програмами лояльності (метод lsCreate). Організатор системи конфігурує новий канал даних для програми лояльності та забезпечує доступ до нього зі своїх вузлів і при необхідності з вузлів власника програми. В даний канал власником програми розміщується транзакційний модуль програми лояльності. Після перевірки коректності функціоналу його підтверджує організатор системи (метод wmsAccept).

Після підключення програми лояльності для її власника відкривається рахунок в криптовалюті. Для можливості його клієнтів користуватися системою він повинен підтримувати його позитивний загальний баланс (параметр balanceTotal). Для цього він повинен обслуговувати в своїй мережі клієнтів інших програм лояльності (обмін бонусів) та/або виконати емісію криптовалюти. Емісія необхідна в тому випадку коли клієнти даної програми лояльності більше користуються іншими програмами в зрівнянні з тим скільки клієнтів інших програм використовують дану програму лояльності. Власник програми лояльності при критичному зниженні балансу свого рахунку за допомогою модулю операцій з криптовалютою (метод emisCreate) емітує необхідний йому об’єм криптовалюти. Організатор системи, після отримання гарантій на дану суму (появи коштів на овердрафт рахунку, отриманні гарантійного листа) виконує підтвердження гарантій емісії (метод emisAccept) та поповнення рахунку програми лояльності на вказану суму.

Користувачі завантажують на свій пристрій мобільний додаток мобільного гаманця. В мобільний гаманець завантажуються цифрові аналоги (токени) карток лояльності програм, в яких клієнт приймає участь. Мобільний гаманець використовує технологію NFC для зв’язку з терміналом точки продажу. Отримавши від терміналу ідентифікатор програми лояльності та параметри покупки модуль керування гаманцями клієнта через облікові модулі програм лояльності інформує користувача про наявні у нього картки лояльності та бонусні кошти на них. За допомогою модулю керування програмами лояльності (метод rateGet) користувач інформується про курси обміну існуючих у нього бонусних балів в бонуси програми лояльності даної торгової точки. Вибрані клієнтом бонуси обмінюються за допомогою транзакційних модулів відповідних програм лояльності та модулю операцій в криптовалюті. Обмін виконується в два етапи: спочатку бонуси, які використовуються переводяться в криптовалюту за курсом продажу програми лояльності, а потім з тимчасового рахунку клієнта в криптовалюті вони зараховуються в програму лояльності точки продажу за курсом покупки точки

продажу. Крім списання та зарахування бонусів на клієнтських рахунках відбувається також списання та зарахування на криптовалютних рахунках відповідних програм лояльності. Після завершення всіх операцій обміну бонусів мобільний гаманець зв'язується з терміналом точки продажу і передає йому дані клієнтської картки лояльності та суму бонусів, яку користувач бажає використати.

В звітний період організатор системи за допомогою модулю розрахунків між організаціями виконує операцію взаєморозрахунків (метод settlementCalc). Даний метод вибирає клієнтські транзакції здійснені від моменту попереднього розрахунку, обчислює за ними комісійні винагороди та кошти які власники програм лояльності повинні сплатити. Кошти можуть бути сплачені за рахунок отриманої криптовалюти від інших програм лояльності (перенесення зобов'язань на іншого учасника), або якщо даних коштів недостатньо – за рахунок погашення емісійних коштів. При погашенні емісійних коштів власнику програми необхідно оплатити їх з гарантійного рахунку фіатними коштами.

2.2. Механізм розгортання блокчейн мережі

Архітектура блокчейн мережі, яка розгортається представлена на рис.2.1.

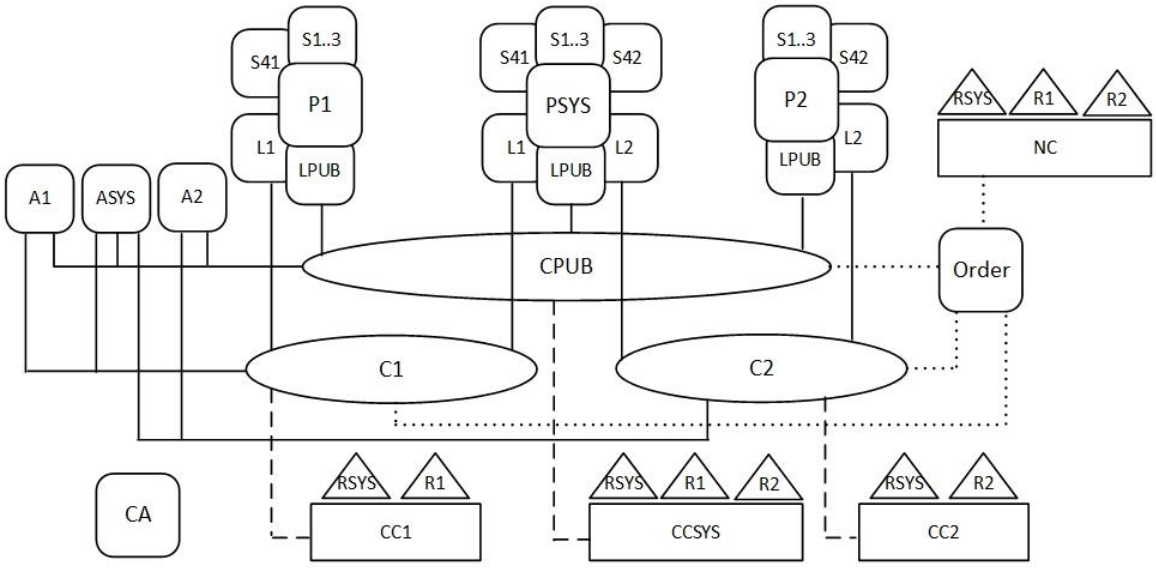


Рисунок 2.1 – Розгортання блокчейн мережі

R1 і R2 являються організаціями, які володіють програмами лояльності, а RSYS – організатор системи. Мережа блокчейн обов'язково розгортається на вузлах організатора PSYS, але додатково, для забезпечення більш довірчих відносин до неї можуть бути включені вузли власників програм лояльності P1 і P2. Доступ власників до блокчейн відбувається за допомогою транзакційних модулів програм лояльності A1 і A2 відповідно, а організатора – модулем керування гаманцями клієнтів PSYS.

Для доступу до публічної інформації (операції з криптовалютою, курси обміну бонусів, взаєморозрахунки) організується канал CPUB, політика роботи якого керується усіма учасниками згідно правил CCSYS. До даного каналу підключаються усі вузли системи, кожен з який веде реєстр ланцюга блоків LPUB. Організатор розміщує в публічному каналі смарт-контракти S1..3:

1. LoyaltySystemContract – керування програмами лояльності та їх рахунками в криптовалюті, курсами обміну бонусів.
2. CryptoCurrencyContract – виконання операцій в криптовалюті (емісія, покупка, продаж, взаєморозрахунки).
3. SettlementContract – виконання процедури взаєморозрахунків між власниками програм лояльності.

Для забезпечення конфіденційності клієнтської інформації різних програм лояльності, для кожної з них створюється окремий приватний канал (C1 і C2 відповідно). Дані канали також створює організатор системи, але політиками роботи їх керують не усі учасники, а тільки організатор і власник відповідної програми лояльності (CC1 і CC2). Дані політики обмежують доступ до каналів для транзакційних модулів A1 і A2, та забезпечують зберігання реєстра ланцюга блоків (L1, L2) тільки на вузлах організатора і власника відповідної програми. Власник програми лояльності розміщує в приватному каналі смарт-контракт TransExternContract (S41, S42) в якому визначено алгоритм обміну бонусів клієнтів (покупки та продажу криптовалюти).

Служба Order забезпечує консенсус роботи в усіх канал блокчейн мережі згідно правил зазначених в конфігурації мережі NC.

Центр сертифікації CA використовується для видачі X.509 сертифікатів для вузлів PSYS, P1, P2 та програмних модулів ASYS, A1 і A2. Процес зіставлення сертифікатів з організатором та власниками програм лояльності виконується сервісом Membership Services Provider (MSP) який конфігурується правилами мережі NC та каналів CCPUB, C1, C2.

Розроблена система для розгортання використовує мережу IBM Blockchain Platform, однак як альтернатива дозволяється для експлуатації використовувати open-source продукт HyperLedger Fabric 2.2.

2.3. Модуль керування програмами лояльності

Основною метою даного модулю являється ведення реєстру програм лояльності та їх рахунків в криптовалюті. Даний модуль представляє собою смарт-контракт на мові програмування Java який розміщується в публічному каналі блокчейн до якого мають доступ усі учасники системи. Використання функціоналу модулю здійснюється підключенням до каналу CPUB та викликом методів контракту “LoyaltySystemContract”. При цьому модуль керування гаманцями клієнта повинен для авторизації використовувати сертифікат організатора системи, а облікові модулі програм лояльності – сертифікати власників відповідних програм лояльності. Перелік методів контракту для взаємодії з модулями системи та зовнішніми програмами наведено у таблиці 2.1.

Таблиця 2.1 – Методи-смарт контракту LoyaltySystemContract

Назва	Доступ	Опис
init	орг.	Ініціалізація модуля
lsCreate	пр.л.	Створення нової програми лояльності
lsGetAll	усі	Отримати список усіх програм лояльності
lsGet	усі	Отримати інформацію про програму лояльності

Кінець таблиці 2.1

Назва	Доступ	Опис
rateChange	пр.л.	Змінити курси обміну бонусів програми лояльності
rateGet	усі	Отримати курс обміну бонусів програми лояльності
balanceEmisChange	орг.	Емісія криптовалюти програмою лояльності після її верифікації організатором
balancesChange	орг.	Утримання/нарахування коштів на рахунок програми лояльності після процедури розрахунків
lockCreate	пр.л.	Блокування коштів для клієнтської транзакції
lockExec	пр.л.	Модифікація рахунку для виконання клієнтської транзакції та зняття блокування
lockCancel	пр.л.	Відміна клієнтської транзакції та блокування коштів

Структура даних LoyaltySystem зберігає в приватній колекції реєстру ланцюга блокчейну інформацію програми лояльності:

- ідентифікатор програми *id* та ідентифікатор власника сертифікату *key*;
- поточний стан рахунку в криптовалюті (обсяг емітованої програмою лояльності валюти *balanceEmis*, обсяг валюти яка в даний момент знаходиться на її рахунку *balanceTotal* та обсяг заблокованих коштів *balanceLock*);
- список незавершених (блокуючих) клієнтських транзакцій *lockList*;
- курси обміну бонусів *rate* (продажу *buy*, покупки *sell* та розмірності бонусів *scale*).

Текст програми модулю наведено в додатку Б.

2.4. Модуль операцій в криптовалюті

Даний модуль забезпечує операції з криптовалютою в системі. Криптовалюта використовується для наступних цілей:

- забезпечення платоспроможності власників програм лояльності (за рахунок овердрафтів або гарантійних листів при емісії ними криптовалюти);
- як розрахункова валюта при обчислені комісійних винагород за клієнтські транзакції;
- для встановлення крос-курсів обміну бонусів між програмами лояльності (замість встановлення прямих курсів між програмами лояльності).

Цей модуль забезпечує виконання транзакцій з криптовалютою та зберігання їх в реєстрі ланцюга блокчейн. При виконанні операцій він взаємодіє з модулем керування програмами лояльності в якому зберігаються криптовалютні рахунки учасників системи.

Даний модуль представляє собою смарт-контракт на мові програмування Java який розміщується в публічному каналі блокчейн до якого мають доступ усі учасники системи. Використання функціоналу модулю здійснюється підключенням до каналу CPUB та викликом методів контракту “CryptoCurrencyContract”. Перелік методів контракту для взаємодії з модулями системи та зовнішніми програмами наведено у таблиці 2.2.

Таблиця 2.2 – Методи смарт-контракту CryptoCurrencyContract

Назва	Доступ	Опис
init	орг.	Ініціалізація модуля
emisCreate	пр.л.	Емісія криптовалюти власником програми лояльності
emisAccept	орг.	Зарахування емісійних коштів організатором після верифікація гарантій емісії
emisReject	орг.	Відхилення організатором емісії
emisCancel	пр.л.	Скасування емісії власником програми лояльності
emisGetWait	орг.	Отримання списку емісій, які очікують верифікації

Кінець таблиці 2.2

Назва	Доступ	Опис
emisGetOne	пр.л. орг.	Отримання інформації про стан емісії
emisGetPeriod	пр.л. орг.	Отримання списку емісій за період вказаною програмою лояльності
settlementPrepareByLs	орг.	Розрахунок комісійної винагороди за клієнтські транзакції вказаної програми лояльності
settlementCreate	орг.	Виконання розрахункової транзакції
settlementGetPeriod	орг.	Отримання списку розрахункових транзакцій за період
buyExec	пр.л.	Зарахування криптовалюти на тимчасовий рахунок клієнта за продаж їм бонусів
buyRefund	орг.	Скасування транзакції продажу
buyGetOne	усі	Отримання інформації про транзакцію продажу бонусів
sellExec	пр.л.	Зарахування криптовалюти на рахунок програми лояльності з тимчасового рахунку клієнта за купівлю їм бонусів
sellRefund	орг.	Скасування транзакції купівлі
sellGetOne	усі	Отримання інформації про транзакцію купівлі бонусів

Модуль оперує з транзакціями чотирьох типів: емісія криптовалюти *TransEmis*, розрахункова *TransSettlement*, продаж бонусів *TransBuy*, купівля бонусів *TransSell*. Вони мають спільні характеристики: унікальний ідентифікатор *id*, ідентифікатор програми лояльності *lsId*, дату створення *dtCreate*, дату виконання *dtExec*, стан *state*, причину відмови/скасування *stateMsg*.

Емісійна транзакція додатково має атрибут суми емісії криптовалюти *amount*.

Розрахункова транзакція має атрибут суми комісії організатора *amount* та деталізацію розрахунків з кожним власником програми лояльності

transSettlementLsList. По кожній програмі лояльності розраховуються наступні показники:

- сума комісії за клієнтські транзакції *amountTotal*;
- частка комісії, яка повинна бути сплачена за рахунок емісійних (гарантійних) коштів payEmis;
- сума коштів з емісії програми лояльності, яку пред'явили до сплати інші програми лояльності payLoan (вона також списується з емісійного рахунку);
- фінальна сума до сплати за період, яка буде виставлена власнику програми лояльності *amountPay*.

Клієнтські транзакції продажу та купівлі бонусів мають наступні додаткові атрибути:

- сума в бонусних одиницях *bonusAmount* та їх розмірність *bonusScale*;
- відповідна бонусам сума в криптовалюті *ccAmount*;
- посилання на транзакцію в зовнішній системі *extrnId*;
- ідентифікатор запису блокування коштів на рахунку програми лояльності *lockId*;
- дата скасування транзакції *dtRefund*;
- стан розрахунку комісії *stateSettlement*.

Крім того транзакція купівлі бонусів містить посилання на транзакцію продажу (тимчасовий рахунок клієнта в криптовалюті) *transBuyId*.

В транзакціях купівлі і продажу відсутня інформація про клієнта, тому доступ до них являється публічним та необмеженим.

Текст програми модулю наведено в додатку В.

2.5. Транзакційний модуль програми лояльності

Даний модуль виконує операції продажу та купівлі за криптовалюту бонусів програми лояльності. Оскільки він взаємодіє з клієнтськими рахунками та зберігає інформацію про них, з метою забезпечення

конфіденційності для нього організується окремий канал блокчейн CCi (де i – ідентифікатор програми лояльності). До даного каналу мають можливість підключення тільки вузли організатора і власника програми лояльності. Доступ до даного каналу дозволено тільки організатору (модуль керування гаманцями клієнтів) та власнику програми лояльності (обліковий модуль відповідної програми лояльності). Власник програми лояльності розміщує в даному каналі смарт-контракт «TransExtrnContract» який реалізує продаж та купівлю бонусів за специфічним для даної програми лояльності алгоритмом. Приклад шаблону такого смарт-контракту на мові Java наведено в додатку Д. Після ініціалізації смарт-контракту власником програми лояльності він переходить в стан «очікує верифікації». Тільки після перевірки його алгоритму організатором системи, який повинен надати згоду (метод wmsAccept) контракт переходить в стан «активний» і можуть виконуватися транзакції продажу та купівлі. В модулі забезпечення консенсусу для даного каналу конфігурується виконання та підтвердження транзакцій тільки вузлами організатора системи та вузлами власника програми лояльності. Для забезпечення коректної роботи модуля керування гаманцями клієнтів та облікового модуля програми лояльності в смарт-контракті обов'язково повинні бути реалізовані методи вказані у таблиці 2.3.

Таблиця 2.3 – Методи смарт-контракту TransExtrnContract

Назва	Доступ	Опис
init	пр.л.	Ініціалізація модуля
wmsAccept	орг.	Верифікація та активізація смарт-контракту організатором системи
buyExtrnExec	пр.л.	Обмін клієнтом своїх бонусів на криптовалюту.
buyExtrnRefund	орг.	Скасування транзакції обміну на криптовалюту.
buyExtrnGetOne	орг. пр.л.	Отримання інформації про транзакцію продажу бонусів

Кінець таблиці 2.3

Назва	Доступ	Опис
sellExec	пр.л.	Обмін клієнтом криптовалюти з тимчасового рахунку на бонуси програми лояльності
sellRefund	орг.	Скасування транзакції купівлі
sellGetOne	орг. пр.л	Отримання інформації про транзакцію купівлі бонусів

Оскільки в операціях задіяні криптовалютний рахунок програми лояльності та тимчасовий рахунок клієнта, даний модуль взаємодіє з модулем операцій в криптовалюті (виклики методу *buyExec* для продажу та методу *sellExec* для купівлі). Усі транзакції зберігаються в реєстрі ланцюгу каналу блокчейна. Кожна транзакція має наступні атрибути:

- унікальний ідентифікатор транзакції *id*;
- посилання на транзакцію в криптавлютному модулі *transCCid*;
- сума в бонусних одиницях *bonusAmount* та їх розмірність *bonusScale*;
- відповідна бонусам сума в криптовалюті *ccAmount*;
- посилання на транзакцію в зовнішній системі *extrnId*;
- дата створення транзакції *dtCreate*;
- дата виконання транзакції *dtExec*;
- дата скасування транзакції *dtRefund*;
- стан транзакції *state*;
- причина скасування транзакції *stateMsg*;
- дані клієнтського бонусного рахунку (ідентифікатор картки) *clientCard*.

Транзакція купівлі має додатковий атрибут посилання на тимчасовий рахунок клієнта в криптовалюті (який відповідає транзакції продажу в криптовалютному модулі) *transBuyId*.

2.6. Модуль розрахунків між організаціями

За виконання клієнтських транзакцій в системі утримується комісія згідно тарифного плану затвердженого для програми лояльності. Обрахунок комісії здійснюється не в режимі онлайн, а при закритті звітного періоду. В звіті враховуються усі транзакції, які не потрапили в попередні періоди (аналізується значення атрибуту транзакції *settlementState*). Комісія береться як частка (процент) від суми транзакції в криптовалюті та/або фіксована сума. Тарифний план затверджується організатором для кожної програми лояльності окремо. Після обрахунку загальної суми комісії, якщо на рахунку власника програми лояльності присутні кошти, емітовані іншими програмами лояльності, то він розраховується ними (зворотній викуп). Решту суми він повинен погасити за рахунок гарантійних коштів на овердрафт рахунку в фіатній валюті (при цьому зменшується об’єм емісії криптовалюти на його рахунку). Крім того гарантійними коштами повинна бути погашена сума, яку йому повернули інші програми лояльності.

Оскільки алгоритм розрахунку повинен бути доступним для ознайомлення усім учасникам – смарт контракт “SettlementContract” розміщується в публічному каналі CPUB. Але оскільки тарифи являються конфіденційною інформацію між організатором і власником програми, вони розміщуються в приватній колекції реєстру ланцюгу блокчейн з обмеженим доступом. Контракт розроблений на мові Java розміщується в блокчейн організатором системи та складається з методів зазначених в таблиці 2.4.

Таблиця 2.4 – Методи смарт-контракту SettlementContract

Назва	Доступ	Опис
init	орг.	Ініціалізація модуля організатором системи
tarifSet	орг.	Встановлення тарифного плану для програми лояльності
tarifGetOne	орг. пр.л	Отримання поточного тарифного плану для програми лояльності

Кінець таблиці 2.4

Назва	Доступ	Опис
tarifGetAll	орг.	Отримання списку тарифних планів для усіх програм лояльності
settlementCalc	орг.	Закриття звітного періоду з розрахунком транзакційних комісій та платежів між організаціями

Кожний тарифний план характеризується наступними атрибутами:

- ідентифікатор програми лояльності lsId;
- процентна частка від суми транзакції продажу prcBuy;
- фіксована сума комісії для транзакції продажу valBuy;
- процентна частка від суми транзакції купівлі prcSell;
- фіксована сума комісії для транзакції купівлі valSell.

2.7. Керування гаманцями клієнтів

Оскільки клієнт являється учасником множини програм лояльності система створює для нього спеціальний профіль (гаманець) в який об’єднуються аккаунти його доступу до облікових модулів програм лояльності (цифрові версії карток лояльності). Для доступу до гаманця для клієнта була розроблена програма «Wallet» на мові Java. Вона працює на пристроях з операційною системою Android 7.0 та вище. Оскільки для платіжних операцій вона контактує з POS терміналом торгової точки за протоколом NFC, пристрій клієнта повинен мати NFC модуль. Для зв’язку з модулем керування гаманцями в момент платіжної операції пристрій клієнта повинен мати активне з’єднання з мережею Інтернет.

Для керування гаманцями клієнтів було розроблено «Wallet Management System». Його розроблено на мові Java з використанням фреймворку Spring 5.3.19. В якості сховища інформації було використано документоорієнтовану СУБД MongoDB 5.0.6. Розгортання елементів

модулю здійснюється з використанням технології контейнеризації Docker та використання серверу кластеризації Kubernetes (Minikube 1.25.2).

Доступ до функціоналу серверу керування гаманцями здійснюється через відкритий розроблений API “Wallet Management System”. Він розроблений у вигляді REST інтерфейсу. Одночасно з розробкою виконувалося документування інтерфейсу згідно специфікації OpenAPI 3.0. Веб інтерфейс Wallet Management System API розміщено на сервері WMS за адресою https://*wms_server*/wms/api.html. Для його візуалізації використано фреймворк Swagger UI, зовнішній вигляд якого представлено на рис. 2.2.

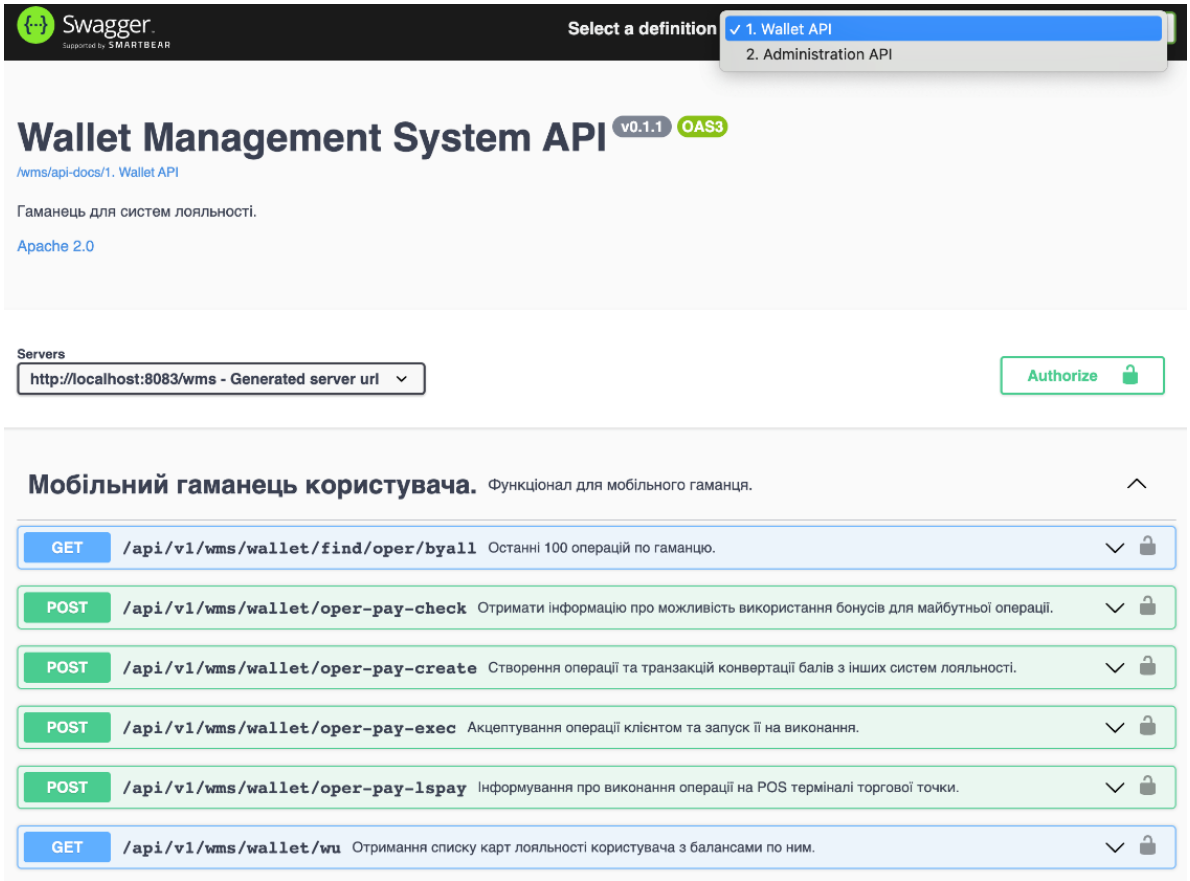


Рис2.2. Wallet Management System API

Використання при розробці REST контролерів бібліотеки springdoc дозволило створити майданчик для розробки і тестування клієнтських мобільних додатків та дозволяє розробникам стороннього програмного забезпечення виконувати інтегрування з системою виконуючи REST запити через веб-інтерфейс, як це показано на рис.2.3.

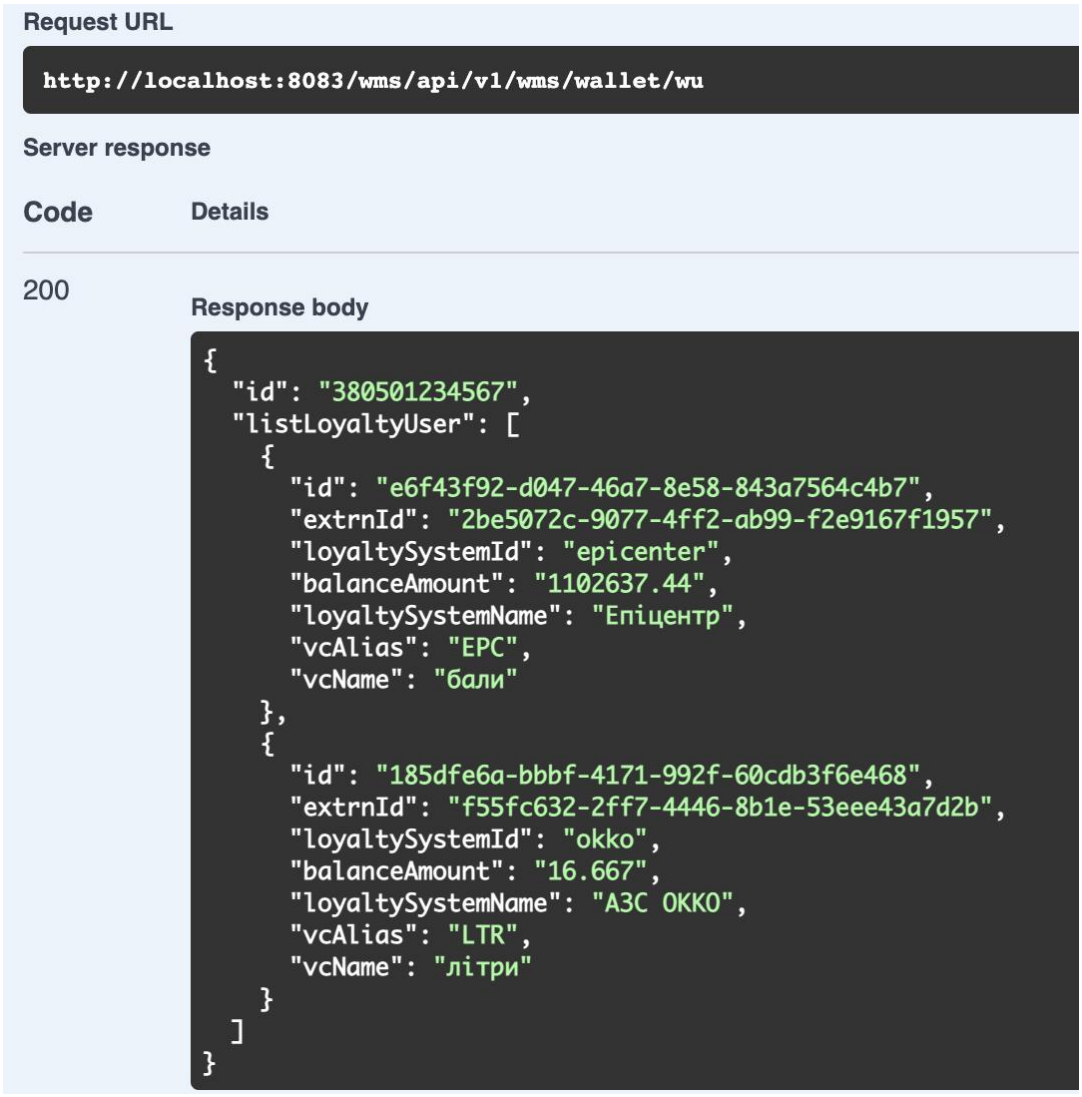


Рис.2.3. Приклад запиту до WMS «Список рахунків гаманця»

Для забезпечення контрольованого доступу до інтерфейсу використовується Bearer аутентифікація з двома типами ролей: адміністратор та клієнт (власник гаманця). Формат авторизаційного токену відповідає специфікації JWT.

2.8. Обліковий модуль програми лояльності

Для взаємодії з зовнішніми системами програм лояльності використовуються спеціалізовані інтеграційні адаптери, які розміщуються на території власника ПЛ. Зовнішня система ПЛ повинна реалізувати API “Loyalty System” з наступними endpoint:

- GET “find/user/bytel/{tel}” – пошук рахунку клієнта за номером телефону;

- GET “lu” – отримання балансу рахунку клієнта;
- POST “lu” – відкриття рахунку (випуск картки) нового клієнта;
- POST “transexternal-replenisment” – виконати транзакцію поповнення бонусів на рахунок клієнта;
- POST “transexternal-lock” – заблокувати кошти на рахунку клієнта для подальшого виконання транзакції списання;
- POST “transexternal-unlock” – відмінити блокування коштів на рахунку клієнта;
- POST “transexternal-withdrawal” – виконати транзакцію списання бонусів з рахунку клієнта;
- GET “find/trans/byuser” – пошук транзакцій по рахунку клієнта.

Адміністрування облікового модулю здійснюється співробітниками власника ПЛ, що забезпечує безпеку згідно з політикою компанії та захищає приватний ключ ПЛ від несанкціонованого доступу.

Модуль розроблено на мові Java з використанням фреймворку Spring 5.3.19. Розгортання елементів модулю здійснюється з використанням технології контейнеризації Docker та використання серверу кластеризації Kubernetes (Minikube 1.25.2).

Доступ модулю керування гаманцями до функціоналу даного адаптеру здійснюється через відкритий розроблений API “Account Management System”. Він розроблений у вигляді REST інтерфейсу. Одночасно з розробкою виконувалося документування інтерфейсу згідно специфікації OpenAPI 3.0. Веб інтерфейс Account Management System API розміщено на сервері AMS за адресою https://*ams_server*/ams/api.html.

Висновок до розділу 2

Використання технології смарт-контрактів дозволило розробити розрахунково-транзакційну систему, яка забезпечує довірчі відносини між учасниками системи та організатором системи. Використані алгоритми забезпечують доступ до функціоналу згідно ролей, вказаних в сертифікатах учасників.

Вибраний тип блокчейну (IBM Blockchain Platform) з приватними каналами захищає інформацію клієнтів окремих ПЛ від доступу інших учасників. Використання криптовалютних рахунків власників ПЛ дозволило забезпечити автоматичний фінансовий моніторинг їх кредитоспроможності (за рахунок емісії ними криптовалюти). Крім того, використання криптовалюти для встановлення кроскурсів обміну між програмами лояльності дозволяє уникнути ведення власниками ПЛ великих за розміром таблиць обміну та забезпечує практично необмежену кількість учасників системи.

Використання при розробці технологій контейнеризації та кластеризації дозволило спростити процес розгортання системи та забезпечує масштабування у разі необхідності. Відкриті REST API інтерфейси модулів WMS та AMS дозволяють використовувати систему зі сторонніми клієнтськими додатками, а використання системи ведення документації OpenAPI значно спрощує таку інтеграцію.

РОЗДІЛ 3. ОБСЛУГОВУВАННЯ КЛІЄНТІВ ПРОГРАМ ЛОЯЛЬНОСТІ

3.1. Ведення гаманця користувача

Для взаємодії клієнтів з системою вони повинні завантажити на свій пристрій мобільний додаток «Wallet». Мобільний додаток зв'язується з сервером авторизації модулю керування гаманцями «WMS» за допомогою протоколу OAuth 2.0 [10]. Якщо на сервері відсутній профіль авторизації який відповідає даному пристрою, користувач проходить процедуру реєстрації вказавши логін та пароль. Ідентифікатор пристрою (IMEI) та номер телефону користувача отримуються мобільним додатком з системних параметрів і додаються до створюємого профілю авторизації. Після успішної авторизації OAuth сервер генерує для мобільного додатку два токена в форматі JWT [11]: access token для доступу до ресурсів модулю WMS та refresh token для автоматичного оновлення авторизаційного токена після закінчення його терміну дії.

Access token складається з заголовку, блоку даних та цифрового підпису. Заголовок містить два атрибути: "typ" – тип токену (завжди має значення "JWT"), "alg" – алгоритм підпису (має значення "HS256", що відповідає HMAC SHA-256). Блок даних містить наступні атрибути:

- "iss" – сервер авторизації (значення "wmsauth");
- "sub" – гаманець клієнта (значення відповідає номеру телефону користувача);
- "iat" – timestamp коли було згенеровано токен доступу;
- "exp" – термін дії токену доступу;
- "role" – для якої ролі було згенеровано токен (значення "wu" – гаманець користувача).

Access token кодується в Base64 та передається в http заголовку (Bearer аутентифікація). Модуль WMS виступає в якості сервісу ресурсів OAuth. Отримавши access token він за допомогою публічного ключа сервера авто-

ризації перевіряє валідність цифрового підпису та коректність полів “iss”, “exp”, “role”.

Для отримання інформації про гаманець мобільний додаток “Wallet” відправляє GET запит “wu” без параметрів до модулю WMS. Модуль WMS з поля “sub” access token отримує ідентифікатор гаманця і перевіряє його наявність в СУБД. Якщо запис присутній в СУБД з нього отримується список карток лояльності даного користувача. Якщо запис відсутній, то здійснюється пошук облікових записів клієнта в усіх програмах лояльності. Для цього в їхні облікові модулі відправляються GET запити “find/user/bytel/{tel}”, де tel – номер телефону клієнта. Якщо клієнт зареєстрований в обліковому модулі програми лояльності, модуль WMS в відповіді отримує ідентифікатор облікового запису (номер картки лояльності) та поточний баланс клієнта в бонусах програми. Модуль WMS після завершення опитування створює в СУБД обліковий запис нового гаманця (primary key слугує номер телефону) та прив’язує до нього усі знайдені картки лояльності. Відповідь WMS до мобільного додатку містить наступні параметри:

- “id” – ідентифікатор гаманця;
- “listLoyaltyUser” – список карток лояльності:
  - “id” – ідентифікатор запису картки в СУБД WMS;
  - “extrnId” – токенізований номер картки лояльності;
  - “loyaltySystemId” – ідентифікатор програми лояльності;
  - “loyaltySystemName” – назва програми лояльності;
  - “balanceAmount” – баланс клієнта в одиницях програми лояльності;
  - “vcAlias” – ідентифікатор бонусних одиниць програми лояльності;
  - “vcName” – назва бонусних одиниць програми лояльності.

У випадку, якщо гаманець клієнта уже існує з СУБД отримується список карток лояльності й перед відправкою результату модуль WMS здійснює оновлення балансів за ними. Для цього він опитує облікові модулі програм лояльності за допомогою GET запиту “lu”. При цьому також використ-

товується механізм OAuth авторизації, але тепер модуль WMS виступає в якості клієнтського додатку, а облікові модулі програм лояльності в якості сервісів ресурсів. Для кожної програми лояльності генерується окремі access та refresh токени. Блок даних access token містить наступні атрибути:

- “iss” – сервер авторизації (значення “wmsauth”);
- “sub” – ідентифікатор облікового запису клієнта в програмі лояльності (зазвичай відповідає номеру картки лояльності);
- “iat” – timestamp коли було згенеровано токен доступу;
- “exp” – термін дії токену доступу;
- “role” – для якої ролі було згенеровано токен (значення “wms” – запит від модулю WMS).

Відповідь від облікового модулю програми лояльності містить наступні атрибути:

- “id” –ідентифікатор облікового запису клієнта в програмі лояльності (повинен збігатися зі значенням атрибуту “sub” з access token);
- “tel” – номер телефону клієнта;
- “loyaltySystemId” – ідентифікатор програми лояльності;
- “loyaltySystemName” – назва програми лояльності;
- “balanceAmount” – баланс клієнта в одиницях програми лояльності;
- “vcAlias” – ідентифікатор бонусних одиниць програми лояльності;
- “vcName” – назва бонусних одиниць програми лояльності.

Отримані дані карток клієнта мобільний додаток “Wallet” виводить у вигляді плиток на екрані “Dashboard” як показано на рис.3.1.

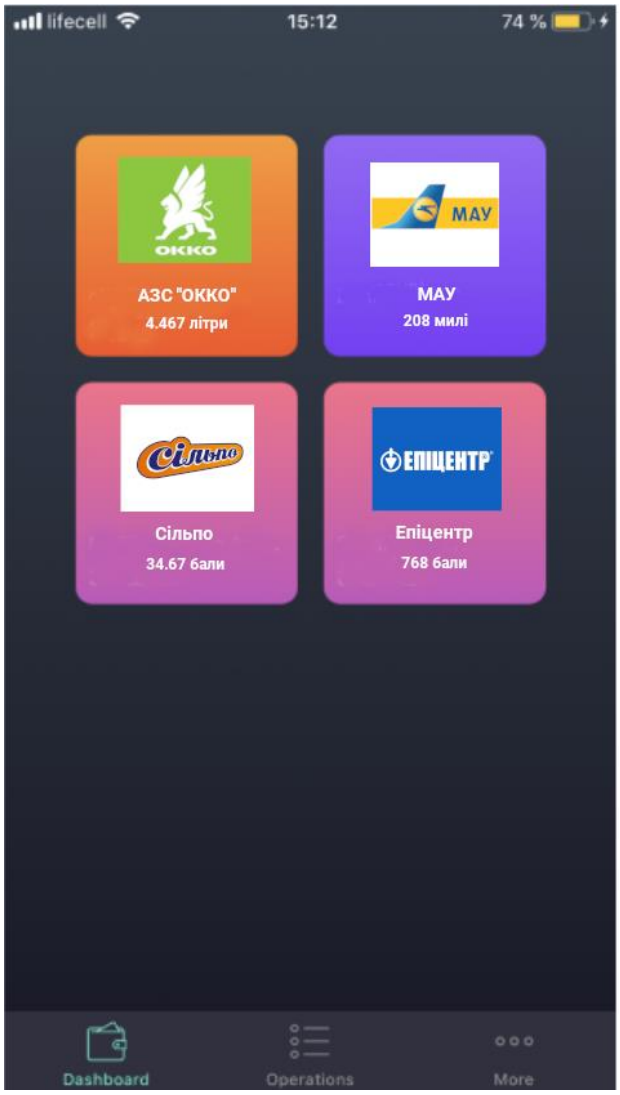


Рисунок 3.1 – Екран “Dashboard” додатку “Wallet”

Для отримання інформації про операції користувачу необхідно перейти на сторінку “Operations”. При цьому “Wallet” здійснює GET запит “find/oper/byall” до WMS без вхідних параметрів. WMS отримавши запит та перевіривши валідність access token на підставі значення атрибуту “sub” здійснює пошук останніх операцій по карткам лояльності прив’язаним до вказаного гаманця. Результат пошуку (останні сто операцій) передається у відповіді WMS в наступній структурі:

- “listOper” – список операцій гаманця:
  - “id” – ідентифікатор операції;
  - “walletUserId” – ідентифікатор гаманця клієнта;

- “operType” – тип операції (pay – платіжна операції в торговій точкі);
- “state” – стан виконання (ok – успіх, err – помилка, cancel - скасовано);
- “stateMsg” – текст помилки або причина скасування операції;
- “operCreateDt” – дата та час створення операції;
- “operExecDt” – дата та час виконання операції;
- “loyaltySystemId” – ідентифікатор ПЛ;
- “loyaltySystemName” – назва ПЛ;
- “loyaltyUserId” – ідентифікатор рахунку клієнта (номер картки) в обліковому модулі ПЛ;
- “operCurrencyType” – тип валюти операції (fc – фіатна);
- “operCurrencyAlias” – ідентифікатор валюти (UAH - гривня);
- “operAmount” – сума операції (загальна вартість товарів);
- “operPurpose” – призначення операції;
- “details” – деталі операції (список додаткових атрибутів):
  - “id” – ідентифікатор параметру (bonusAmountIn – бонусний баланс до операції, bonusAmountOut – бонусний баланс після операції, payAmount – сплачено фіатними коштами);
  - “v” – значення параметру;
- “listTrans” – список транзакцій даної операції:
  - “id” – ідентифікатор транзакції;
  - “operId” – ідентифікатор операції;
  - “transType” – тип транзакції (lsExch – обмін бонусів);
  - “state” – стан виконання (ok – успіх, err – помилка, cancel - скасовано);
  - “stateMsg” – текст помилки або причина скасування транзакції;

- “transCreateDt” – дата та час створення транзакції;
- “transExecDt” – дата та час виконання транзакції;
- “loyaltySystemId” – ідентифікатор основної ПЛ транзакції;
- “loyaltySystemName” – назва основної ПЛ транзакції;
- “loyaltyUserId” – ідентифікатор рахунку клієнта (номер картки) в обліковому модулі основної ПЛ (з якої списуються бонуси для транзакції обміну);
- “sendCurrencyType” – тип валюти списання (vc - бонуси);
- “sendCurrencyAlias” – ідентифікатор бонусів ПЛ списання;
- “sendAmount” – сума списуємих бонусів з картки лояльності ПЛ списання;
- “rcptCurrencyType” – тип валюти зарахування (vc - бонуси);
- “rcptCurrencyAlias” – ідентифікатор бонусів ПЛ зарахування;
- “rcptAmount” – сума зараховуємих бонусів на карту клієнта ПЛ зарахування;
- “transPurpose” – назва транзакції;
- “details” – деталі транзакції (список додаткових атрибутів):
    - “id” – ідентифікатор параметру (sendTransExtrnId – ідентифікатор в транзакційному модулі ПЛ списання, rcptTransExtrnId – ідентифікатор в транзакційному модулі ПЛ зарахування);
    - “v” – значення параметру.

Приклад відповіді WMS наведено на рис.3.2.

```
{
  "listOper": [
    {
      "id": "c0e9b4ea-5d58-46ba-9ceb-9520a3d99ce0",
      "walletUserId": "380501234567",
      "operType": "pay",
      "state": "ok",
      "stateMsg": null,
      "operCreateDt": "2022-02-07T23:26:20.579",
      "operExecDt": "2022-02-07T23:27:29.438",
      "loyaltySystemId": "okko",
      "loyaltySystemName": "АЗС ОККО",
      "loyaltyUserId": "185dfe6a-bbbf-4171-992f-60cdb3f6e468",
      "operAmount": "10000.00",
      "operCurrencyType": "fc",
      "operCurrencyAlias": "UAH",
      "operPurpose": "Оплата бензину.",
      "details": [
        {
          "id": "bonusAmountIn",
          "v": "32.026"
        },
        {
          "id": "bonusAmountOut",
          "v": "16.667"
        },
        {
          "id": "payAmount",
          "v": "9039.22"
        }
      ],
      "listTrans": [
        {
          "id": "fa2db6be-bbba-481a-8215-7a5809cffd05",
          "operId": "c0e9b4ea-5d58-46ba-9ceb-9520a3d99ce0",
          "walletUserId": "380501234567",
          "transType": "lsExch",
          "state": "ok",
          "stateMsg": null,
          "transCreateDt": "2022-02-07T23:26:20.587",
          "transExecDt": "2022-02-07T23:27:29.438",
          "loyaltySystemId": "epicenter",
          "loyaltySystemName": "Епіцентр",
          "loyaltyUserId": "e6f43f92-d047-46a7-8e58-843a7564c4b7",
          "sendAmount": "1000.00",
          "sendCurrencyType": "vc",
          "sendCurrencyAlias": "EPC",
          "rcptAmount": "32.026",
          "rcptCurrencyType": "vc",
          "rcptCurrencyAlias": "LTR",
          "transPurpose": "Обмін EPC[Епіцентр] -> LTR[АЗС ОККО]",
          "details": [
            {
              "id": "sendTransExtrnId",
              "v": "351ee228-ee0b-4e44-91cc-9f755c432392"
            },
```


Рисунок 3.2 – Відповідь WMS “find/oper/byall”

Мобільний додаток виводить отриманий список операцій у вигляді, зображеному на рис.3.3.

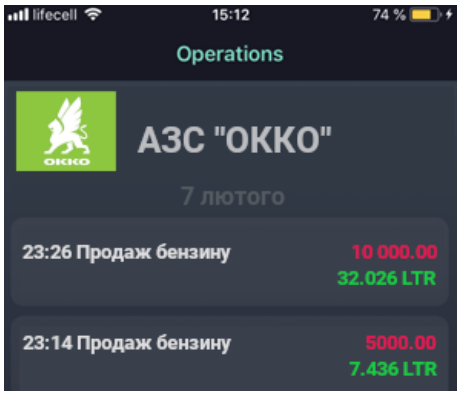


Рисунок 3.3 – Відображення списку операцій в “Wallet”

При виборі окремої операції, “Wallet” відображає детальну інформацію про операцію, у вигляді, зображеному на рис.3.4.

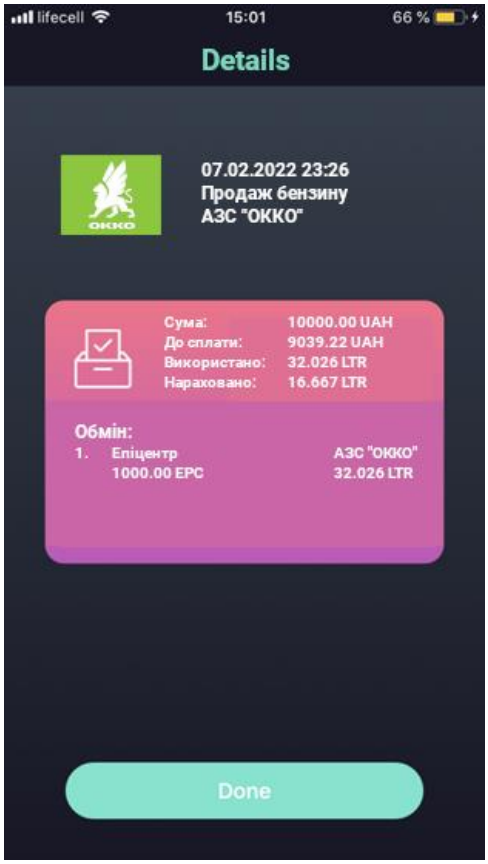


Рисунок 3.4 – Детальна інформація операції в “Wallet”

3.2. Виконання платіжної транзакції

Використання бонусів платіжної програми здійснюється через платіжний термінал торгівельної точки та мобільний додаток “Wallet” клієнта. Алгоритм платіжної транзакції представлено на схемі ІАЛЦ.467200.006 Д3.

Касовий модуль надсилає платіжному терміналу загальну вартість товарів у чеку в фіатній валюті (гривні). Після цього термінал переходить в режим очікування картки лояльності. Користувач розблоковує свій смартфон та підносить його до платіжного терміналу. Додаток “Wallet” після встановлення на телефон встановлює сервіс “NFC Card Emulation” який підтримує обмін даними між двома NFC пристроями згідно специфікації ISO/IEC 14443-4 [12]. Після встановлення каналу зв’язку сервіс мобільного додатку “Wallet” виступає в ролі хоста, а платіжний термінал в ролі серверу. На прикладному рівні взаємодія здійснюється

відправкою “Wallet” APDU команд, а термінал надсилає у відповідь APDU response згідно специфікації ISO/IEC 7816-4 [13].

APDU команда складається з заголовку та опціонального блоку даних.

Заголовок містить наступні поля:

- CLA (1 байт) – клас до якого належить команда;
- INS (1 байт) – визначає команду в межах класу;
- P1 (1 байт) та P2 (1 байт) – параметри команди.

Блок даних складається з:

- LC (1 байт) – довжина опціонального блоку додаткових параметрів;
- Data – додаткові параметри;
- LE (1 байт) – максимальна довжина блоку додаткових параметрів у відповіді на команду.

APDU response складається з:

- Data – додаткових параметрів відповіді;
- SW1 (1 байт) та SW2 (1 байт) – код обробки команди (0x9000 – успіх, інакше - помилка).

На першому етапі взаємодії “Wallet” отримує від платіжного терміналу ідентифікатор програми лояльності виконавши команду “SELECT AID” як показано на рис.3.5.

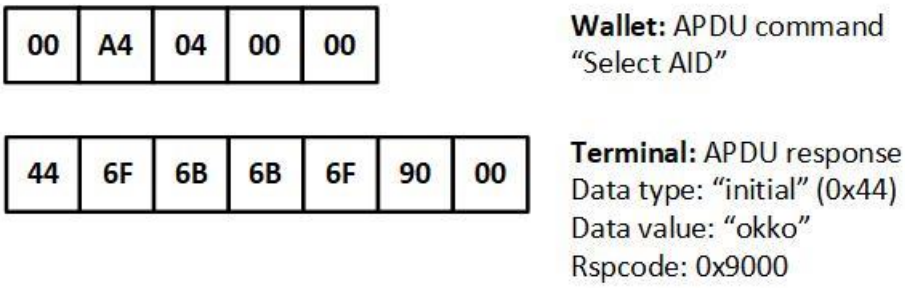


Рисунок 3.5 – Ідентифікація ПЛ у торговій точці.

Наступним кроком, “Wallet” отримує від терміналу суму чеку обраних товарів (рис.3.6) виконавши команду “GET DATA Card Data” згідно специфікації ISO/IEC 7816-6 [14].

Wallet: APDU command
"GET DATA Card Data"

80	CA	00	66	00

Terminal: APDU response
Data type: "Card Data" (0x66)
Tag: "Amount, Authorised, Exp" (0x5F32), "2"
Tag: "Amount, Authorised" (0x9F02), "548.37"
Tag: "Amount, Other, Exp" (0x5F33), "3"
Tag: "Amount, Other" (0x9F03), "18.279"
Rspcode: 0x9000

66	5F	32	02	9F	02	00	00	00	05	48	37

5F	33	03	9F	03	00	00	00	01	82	79	90	00

Рисунок 3.6 – Отримання в терміналу суми чеку

Відповідь терміналу містить в собі суму чеку (Amount, Authorised) та максимальну кількість бонусів ПЛ, які можна використати при оплаті (Amount, Other).

Отже в тестовому прикладі ми отримали від платіжного терміналу, що клієнт знаходиться в торговій точці з програмою лояльності "okko" та бажає здійснити покупку на 548.37 грн в якій можна використати до 18.279 LTR.

Наступним кроком, "Wallet" виконує запит до WMS для отримання актуального стану гаманця клієнта (див. розділ 3.1). Потім виконуються запити для отримання курсів продажу та купівлі для програми лояльності торгової точки та ПЛ з ненульовими балансами гаманця клієнта. Для цього використовуються GET запит "ls/{lsId}/rate". Після отримання запиту модуль керування гаманцями звертається до модулю керування системами лояльності (смарт-контракт "LoyatySystemContract", метод "rateGet") і отримує від нього значення курсу продажу (buy) та купівлі (sell). Відповідь клієнтському додатку відправляється в форматі JSON і містить наступні атрибути:

"loyaltySystemId" – ідентифікатор ПЛ;

"rateBuy" – курс продажу до криптовалюти;

"rateSell" – курс купівлі до криптовалюти.

Для тестового прикладу, дані отримані від WMS наведені в таблиці 3.1

Таблиця 3.1 – Гаманець клієнта до виконання операції

Програма лояльності	Баланс	Купівля	Продаж
okko	1.054	2940	3060
mau	20	1960	2040
silpo	2345	0.98	1.02
epicenter	67.08	96	104

Отримавши інформацію від платіжного терміналу і модулю керування гаманцями, клієнтський додаток “Wallet” виводить її на екран смартфону та дозволяє клієнту вибрати бонуси, які він використає при здійснені платежу. В тестовому прикладі, показаному на рис.3.7 клієнт вибрав для використання бонуси ПЛ “okko”, “mau” та “silpo”.

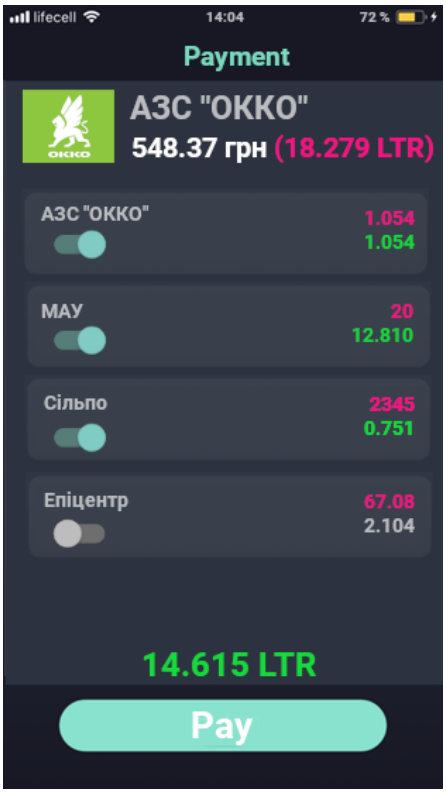


Рисунок 3.7 – Вибір бонусів для платіжної транзакції

Після вибору клієнтом бонусів і натискання ним кнопки “Pay” додаток “Wallet” здійснює POST запит “oper-pay-create” до WMS на створення платіжної операції та обміну бонусів. Запит має наступну структуру:

- “loyaltySystemId” – ідентифікатор ПЛ торгової точки (куди будуть зараховуватися бонуси);
- “listLoyaltyUserPay” – список вибраних бонусів для обміну:

- “loyaltyUserId” – ідентифікатор клієнтського рахунку в ПЛ з якої будуть списуватися бонуси;
- “sendAmount” – сума бонусів для обміну з вказаного рахунку;

- “operAmount” – сума чеку в фіатний валюті (гривні);
- “operPurpose” – назва операції.

Приклад запиту приведено на рис.3.8.


```
{
  “loyaltySystemId”: “okko”,
  “listLoyaltyUserPay”: [
    {
      “loyaltyUserId”: “54b93041-cc3e-47b6-8d9c-c5ac993e5253”,
      “sendAmount”: “20”
    },
    {
      “loyaltyUserId”: “509587cf-9283-4a89-b5ae-69ea5d69541b”,
      “sendAmount”: “2345”
    }
  ],
  “operAmount”: “548.37”,
  “operPurpose”: “Оплата товару”
}
```


Рисунок 3.8 – Запит від “Wallet” на створення платіжної операції

В тестовому прикладі на обмін в ПЛ “ОККО” передаються 20 MIL з картки “МАУ” та 2345 SLP з картки “Сільпо”. Вибрані 1.054 LTR з картки “ОККО” не передаються, оскільки вони не потребують обміну.

Після отримання запиту, модуль керування гаманцями створює платіжну операцію та по кожній з вказаних в запиті карток лояльності виконує запити до облікових модулів відповідних ПЛ на продаж бонусів за криптовалюту (POST запит “trans-buy-create” API “Acount Management System”). Ідентифікатор рахунку клієнту включається в поле “sub” згенерованого access token, а сам запит має наступну структуру:

- “extrnId” – ідентифікатор транзакції обміну в WMS;
- “bonusAmountChange” – сума бонусів для продажу;
- “purpose” – назва операції.

Приклад продажу 20 MIL з картки лояльності “МАУ” приведено на рис.3.9.

```
{
  “extrnId”: “e6743f92-d047-46a7-8e58-843a7564d587”,
  “bonusAmountChange”: “20”,
  “purpose”: “Обмін для АЗС ОККО”
}
```


Рисунок 3.9 – Запит на продаж бонусів з картки лояльності

Обліковий модуль виконує запит “transexternal-lock” до API “LoyaltySystem” на блокування вказаної суми бонусів на рахунку клієнта. Після успішного блокування виконується звернення до транзакційного модуля ПЛ (метод “buyExtrnExec” смарт-контракту “TransExtrnContract” відповідної ПЛ) на виконання транзакції продажу бонусів за криптовалюту.

Транзакційний модуль ПЛ виконує запит до модулю керування програмами лояльності для отримання поточного курсу продажу (метод “lsGet” смарт-контракту “LoyatySystemContract”). Після отримання курсу розраховується сума в криптовалюті, яка буде зарахована на тимчасовий рахунок клієнта.

Зарахування здійснюється модулем операцій в криптовалюті (метод “buyExec” смарт-контракту “CryptoCurrencyContract”). Даний метод спочатку виконує блокування відповідної суми на рахунку власника ПЛ виконавши запит “lockCreate” смарт-контракту “LoyaltySystemContract”. Блокування відбувається успішно, якщо сума блокування менша або дорівнює сумі загального балансу рахунку та сумі поточних блокувань ПЛ. Це гарантує платоспроможність власника ПЛ, оскільки дана сума не перевищує його гарантійних обов’язків (емісії) та коштів які він може отримати від повернення криптовалюти, емітованої іншими учасниками системи. Отримавши ідентифікатор блокування, модуль операцій в криптовалюті створює транзакцію продажу з відкриттям тимчасового рахунку клієнта і зарахуванням на нього криптовалюти. Потім виконується запит до модулю керування програмами лояльності для зняття коштів з рахунку власника ПЛ (метод “lockExec” смарт-контракту “LoyaltySystemContract”). Приклади операцій з криптовалютними рахунками показано на рис.3.10 (продаж бонусів “МАУ”) та рис.3.11 (продаж бонусів “Сільпо”).

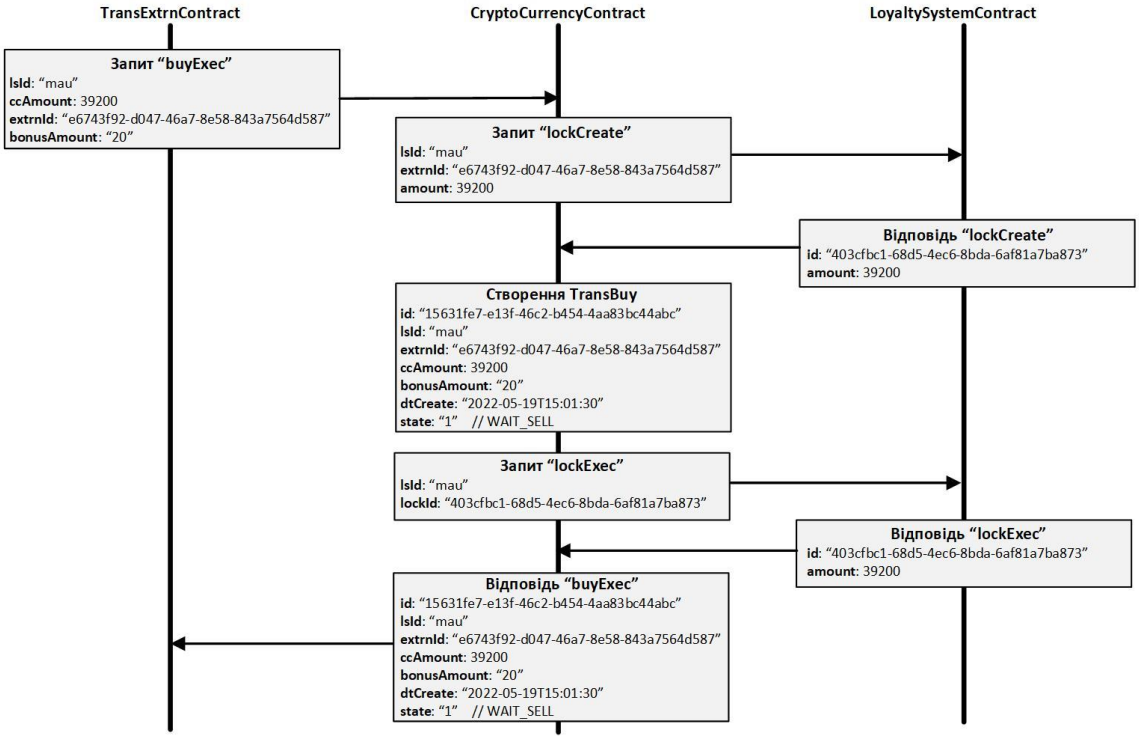


Рисунок 3.10 – Криптовалютна транзакція продажу ПЛ "МАУ"

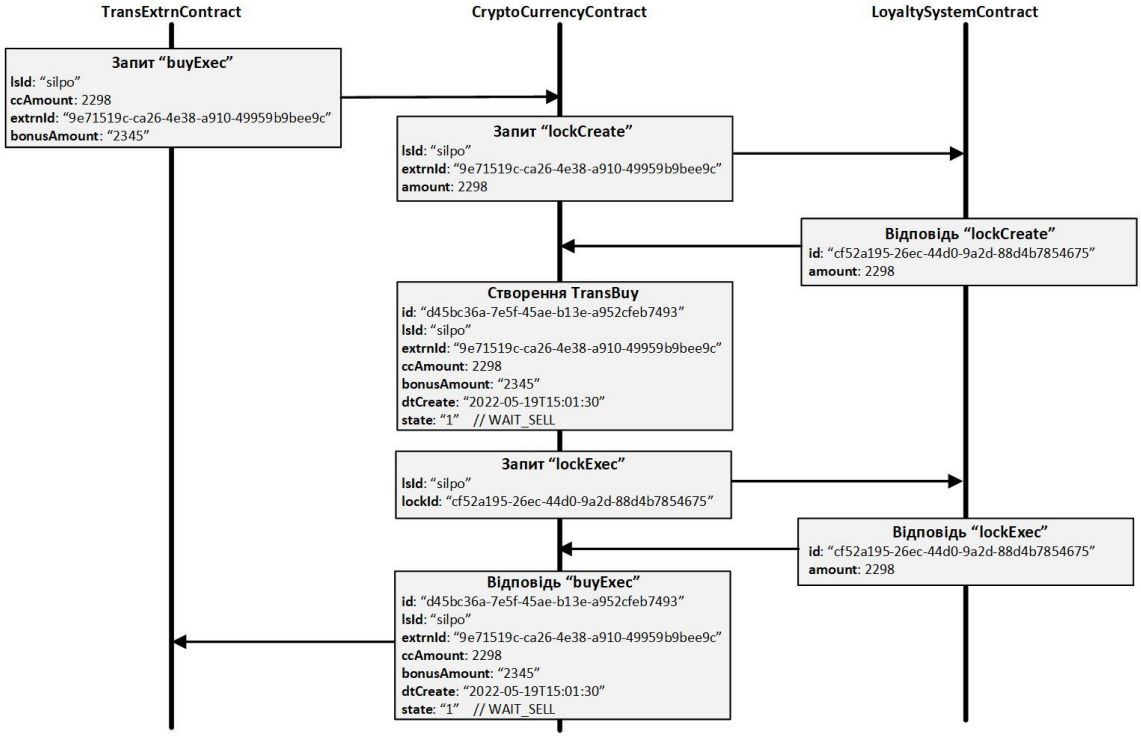


Рисунок 3.11 – Криптовалютна транзакція продажу ПЛ "Сільпо"

Після отримання відповіді від модулю операцій в криптовалюті, транзакційний модуль створює та зберігає в приватній колекції транзакцію продажу бонусів. Приклад виконання продажу бонусів в ПЛ "МАУ" і ПЛ "Сільпо" наведено в таблиці 3.2.

Таблиця 3.2 – Продаж бонусів в ПЛ “МАУ” і ПЛ “Сільпо”

ПЛ	Запит	Відповідь
МАУ	**extrnId:** “e6743f92-d047-46a7-8e58-843a7564d587” bonusAmount: “20” **clientCard:** “54b93041-cc3e-47b6-8d9c-c5ac993e5253”	**id:** “a628aacc-adc2-4bdd-bb5c-15cac6e674f6” bonusAmount: “20” **clientCard:** “54b93041-cc3e-47b6-8d9c-c5ac993e5253” **transCCid:** “15631fe7-e13f-46c2-b454-4aa83bc44abc” ccAmount: 39200 **state:** 2 // OK
Сільпо	**extrnId:** “9e71519c-ca26-4e38-a910-49959b9bee9c” bonusAmount: “2345” **clientCard:** “509587cf-9283-4a89-b5ae-69ea5d69541b”	**id:** “fc47d0a3-c7f3-4b4c-9b8e-72b0c6845857” bonusAmount: “2345” **clientCard:** “509587cf-9283-4a89-b5ae-69ea5d69541b” **transCCid:** “9e71519c-ca26-4e38-a910-49959b9bee9c” ccAmount: 2298 **state:** 2 // OK

Відповідь від транзакційного модуля містить ідентифікатор транзакції продажу (“id”), ідентифікатор тимчасового рахунку клієнта в криптовалюті (“transCCid”) та суму в криптовалюті, отриману від продажу бонусів (“ccAmount”).

Після успішного виконання транзакції продажу, обліковий модуль виконує запит “transexternal-withdrawal” API “LoyaltySystem” який списує з бонусного рахунку клієнта заблоковану суму. Якщо транзакція продажу не була виконана успішно – обліковий модуль знімає блокування з коштів рахунку клієнта (запит “transexternal-unlock” API “LoyaltySystem”).

Відповідь облікового модулю на запит WMS містить наступні атрибути:

- “id” – ідентифікатор транзакції продажу бонусів;
- “extrnId” – ідентифікатор транзакції обміну в WMS;
- “loyaltyUserId” – ідентифікатор бонусного рахунку клієнта;
- “bonusAmount” – сума проданих бонусів;

- “transCCid” – ідентифікатор тимчасового рахунку в криптовалюті клієнта;
- “ccAmount” – сума отриманої криптовалюти;
- “dtCreate” – дата та час створення транзакції;
- “dtExec” – дата та час виконання транзакції;
- “state” – стан транзакції (“ok” – успіх, “err” – помилка);
- “stateMsg” – текст помилки.

Приклад відповіді на запит продажу 20 MIL ПЛ “МАУ” (див. рис.3.9) приведено на рис.3.12.


```
{
  “id”: “a628aacc-adc2-4bdd-bb5c-15cac6e674f6”,
  “extrnId”: “e6743f92-d047-46a7-8e58-843a7564d587”,
  “loyaltyUserId”: “54b93041-cc3e-47b6-8d9c-c5ac993e5253”,
  “bonusAmount”: “20”,
  “transCCid”: “9e71519c-ca26-4e38-a910-49959b9bee9c”,
  “ccAmount”: 39200,
  “dtCreate”: “2022-05-19T15:01:30”,
  “dtExec”: “2022-05-19T15:01:30”,
  “state”: “ok”,
}
```


Рисунок 3.12 – Відповідь облікового модулю на запит продажу бонусів

Модуль керування гаманцями клієнтів перевіряє наявність у клієнта відкритого рахунку в ПЛ торгової точки. У випадку його відсутності виконується POST запит “lu” API “Account Management System” до облікового модулю ПЛ торгової точки з наступними параметрами:

- “walletUserId” – ідентифікатор гаманця клієнта;
- “tel” – номер телефону клієнта;
- “profile” – список атрибутів профілю клієнта:
  - “id” – ідентифікатор атрибуту (name, tel, addr);
  - “v” – значення атрибуту.

Модуль керування гаманцями клієнтів після отримання успішної відповіді на проведення транзакції продажу відправляє POST запит “trans-sell-create” до API “Account Management System” облікового модулю програми лояльності точки продажу. Ідентифікатор рахунку клієнту включається в поле “sub” згенерованого access token, а сам запит має наступну структуру:

- “extrnId” – ідентифікатор транзакції обміну в WMS;
- “transCCid” – ідентифікатор тимчасового рахунку клієнта на який була зарахована криптовалюта при транзакції продажу;
- “ccAmount” – сума криптовалюти, яка використовується для купівлі;
- “purpose” – назва операції.

Приклад купівлі бонусів “АЗС ОККО” за криптовалюту отриману від продажу 20 MIL з картки лояльності “МАУ” (див. рис. 3.12) приведено на рис.3.13.

```
{
  “extrnId”: “e6743f92-d047-46a7-8e58-843a7564d587”,
  “transCCid”: “9e71519c-ca26-4e38-a910-49959b9bee9c”,
  “ccAmount”: 39200,
  “purpose”: “Зарахування від продажу бонусів МАУ”
}
```

Рисунок 3.13 – Запит до облікового модулю на купівлю бонусів

Обліковий модуль отримавши запит на купівлю бонусів виконує запит до модулю керування системами лояльності (метод “lsGet” смартконтракту “LoyaltySystemContract”) для отримання поточного курсу купівлі. Після обрахування суми бонусів, які необхідно зарахувати на рахунок клієнт обліковий модуль виконує запит до транзакційного модулю відповідної ПЛ для виконання транзакції купівлі бонусів за криптовалюту (метод “sellExtrnExec” смарт-контракту “TransExtrnContract”).

Переказ криптовалютних коштів з тимчасового рахунку клієнта на рахунок власника ПЛ здійснюється модулем операцій в криптовалюті (метод “sellExec” смарт-контракту “CryptoCurrencyContract”). Даний метод спочатку перевіряє наявність коштів на тимчасовому рахунку клієнта (атрибут транзакції продажу state=1). Потім виконується блокування відповідної суми з від’ємним знаком на рахунку власника ПЛ (запит “lockCreate” смарт-контракту “LoyaltySystemContract”). Після отримання ідентифікатора блокування, модуль операцій в криптовалюті створює транзакцію купівлі. Потім виконується запит до модулю керування програмами лояльності для зарахування коштів на рахунок власника ПЛ (метод “lockExec” смарт-контракту “LoyaltySystemContract”). Приклади

операцій купівлі з криптовалютними рахунками показано на рис. 3.14 (за продаж бонусів “МАУ”) та рис.3.15 (за продаж бонусів “Сільпо”).

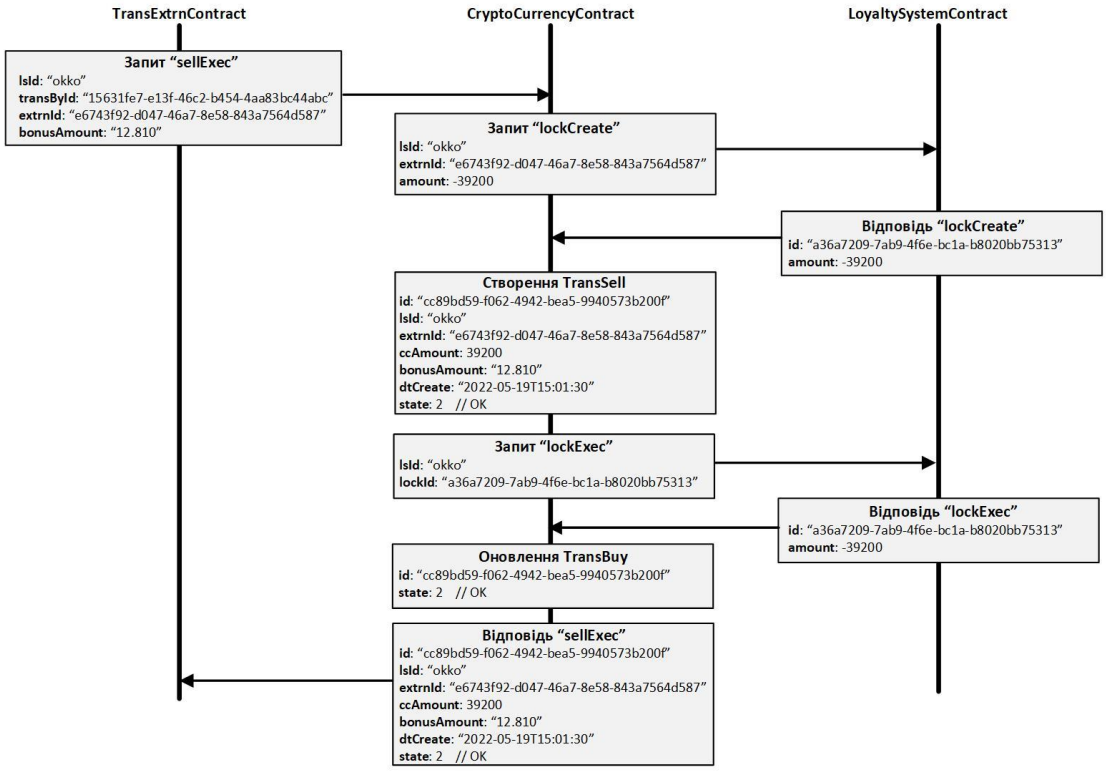


Рисунок 3.14 – Криптовалютна транзакція купівлі (від продажу бонусів “МАУ”)

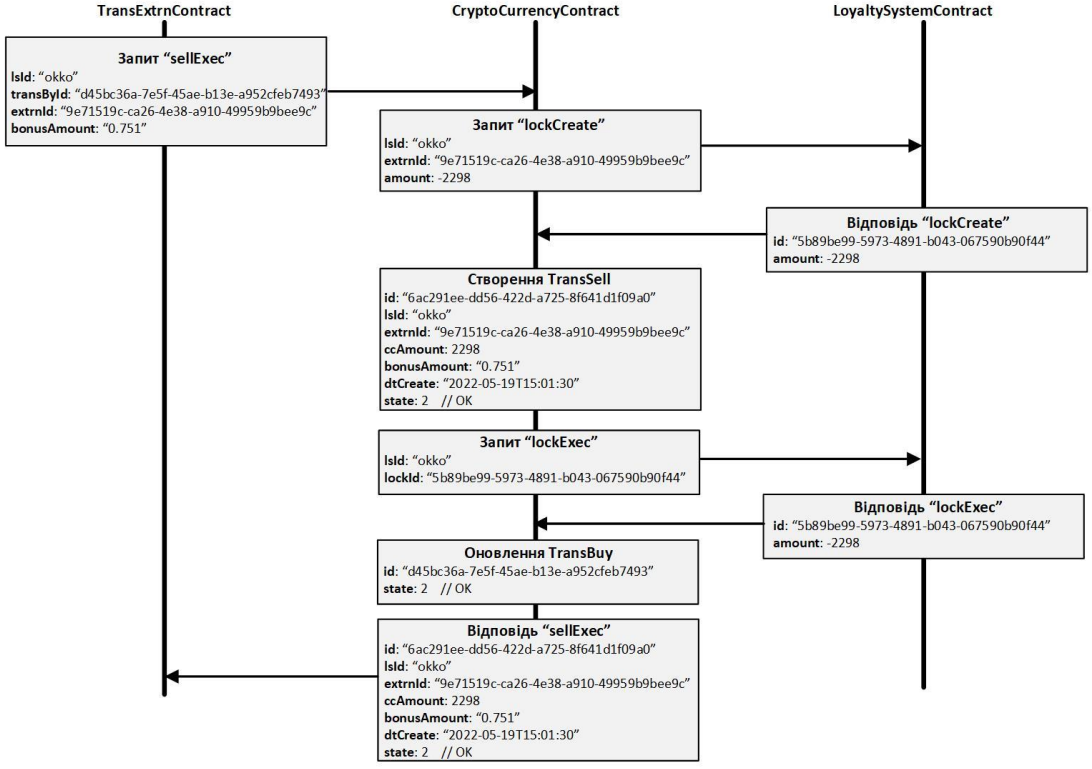


Рисунок 3.15 – Криптовалютна транзакція купівлі (від продажу бонусів “Сільпо”)

Після отримання відповіді від модулю операцій в криптовалюті, транзакційний модуль створює та зберігає в приватній колекції транзакцію купівлі бонусів. Приклад виконання купівлі бонусів ПЛ “АЗС ОККО” за продаж в ПЛ “МАУ” і ПЛ “Сільпо” наведено в таблиці 3.3.

Таблиця 3.3 – Купівля бонусів “АЗС ОККО” за продаж в ПЛ “МАУ” і ПЛ “Сільпо”

ПЛ	Запит	Відповідь
МАУ	**extrnId:** “e6743f92-d047-46a7-8e58-843a7564d587” bonusAmount: “12.810” **transBuyId:** “15631fe7-e13f-46c2-b454-4aa83bc44abc” **clientCard:** “185dfe6a-bbbf-4171-992f-60cdb3f6e468”	**id:** “1cfb8087-f9fe-4f61-9ea8-69d2b4b90c6f” bonusAmount: “12.810” **clientCard:** “185dfe6a-bbbf-4171-992f-60cdb3f6e468” **transCCid:** “cc89bd59-f062-4942-bea5-9940573b200f” ccAmount: 39200 **state:** 2 // OK
Сільпо	**extrnId:** “9e71519c-ca26-4e38-a910-49959b9bee9c” bonusAmount: “0.751” **transBuyId:** “9e71519c-ca26-4e38-a910-49959b9bee9c” **clientCard:** “185dfe6a-bbbf-4171-992f-60cdb3f6e468”	**id:** “004ac897-078e-419e-a082-feda9e610f22” bonusAmount: “0.751” **clientCard:** “185dfe6a-bbbf-4171-992f-60cdb3f6e468” **transCCid:** “6ac291ee-dd56-422d-a725-8f641d1f09a0” ccAmount: 2298 **state:** 2 // OK

Відповідь від транзакційного модуля містить ідентифікатор транзакції купівлі (“id”), ідентифікатор транзакції у криптовалюті (“transCCid”), суму в криптовалюті, яку витратили на купівлю бонусів (“ccAmount”) та сума отриманих бонусів (“bonusAmount”).

Після успішного виконання транзакції купівлі, обліковий модуль виконує запит “transexternal-replenisment” API “LoyaltySystem” який поповнює бонусний рахунок клієнта на суму “bonusAmount”.

Відповідь облікового модулю на запит WMS містить наступні атрибути:

- “id” – ідентифікатор транзакції купівлі бонусів;
- “extrnId” – ідентифікатор транзакції обміну в WMS;
- “loyaltyUserId” – ідентифікатор бонусного рахунку клієнта;

- “bonusAmount” – сума отриманих бонусів;
- “transCCid” – ідентифікатор транзакції у криптовалюті;
- “ccAmount” – сума витраченної криптовалюти;
- “dtCreate” – дата та час створення транзакції;
- “dtExec” – дата та час виконання транзакції;
- “state” – стан транзакції (“ok” – успіх, “err” – помилка);
- “stateMsg” – текст помилки.

Приклад відповіді на запит купівлі від продажу 20 MIL ПЛ “МАУ” (див. рис.3.13) приведено на рис.3.16.


```
{
  “id”: “1cfb8087-f9fe-4f61-9ea8-69d2b4b90c6f”,
  “extrnId”: “e6743f92-d047-46a7-8e58-843a7564d587”,
  “loyaltyUserId”: “185dfe6a-bbbf-4171-992f-60cdb3f6e468”,
  “bonusAmount”: “12.810”,
  “transCCid”: “cc89bd59-f062-4942-bea5-9940573b200f”,
  “ccAmount”: 39200,
  “dtCreate”: “2022-05-19T15:01:30”,
  “dtExec”: “2022-05-19T15:01:30”,
  “state”: “ok”,
}
```


Рисунок 3.16 - Відповідь облікового модулю на запит купівлі бонусів

Після обробки усього списку на обмін бонусів, модулів WMS відправляє відповідь мобільному додатку “Wallet”, скорочений приклад якої приведено на рис.3.17.


```
{
  “id”: “6cbad63a-5925-4edb-8815-1f9e8bffba71”,
  “loyaltySystemId”: “okko”,
  “loyaltyUserId”: “185dfe6a-bbbf-4171-992f-60cdb3f6e468”,
  “operAmount”: “548.37”,
  “operCurrencyAlias”: “UAH”,
  “listTrans”: [
    {
      “id”: “e6743f92-d047-46a7-8e58-843a7564d587”,
      “loyaltySystemId”: “mau”,
      “loyaltyUserId”: “54b93041-cc3e-47b6-8d9c-c5ac993e5253”,
      “sendAmount”: “20”,
      “sendCurrencyAlias”: “MIL”,
      “rcptAmount”: “12.810”,
      “rcptCurrencyAlias”: “LTR”,
    },
    {
      “id”: “9e71519c-ca26-4e38-a910-49959b9bee9c”,
      “loyaltySystemId”: “silpo”,
      “loyaltyUserId”: “509587cf-9283-4a89-b5ae-69ea5d69541b”,
      “sendAmount”: “2345”,
      “sendCurrencyAlias”: “SLP”,
      “rcptAmount”: “0.751”,
      “rcptCurrencyAlias”: “LTR”,
    }
  ]
}
```


Рисунок 3.17 – Скорочена відповідь WMS на виконання платіжної транзакції

Відповідь WMS містить наступні основні атрибути (повний список знаходиться за адресою https://wms_сервер/wms/api.html):

- “id” – ідентифікатор платіжної операції;
- “loyaltySystemId” – ідентифікатор ПЛ торгової точки;
- “loyaltyUserId” – картка лояльності клієнта ПЛ торгової точки;
- “operAmount” – сума чеку;
- “listTrans” – список транзакцій обміну бонусів:
  - “id” – ідентифікатор транзакції обміну;
  - “loyaltySystemId” – ідентифікатор ПЛ, в якій списуються бонуси;
  - “loyaltyUserId” – картка клієнта, з якої списуються бонуси;
  - “sendAmount” – сума бонусів, які були списані;
  - “rcptAmount” – сума бонусів, які були зараховані.

Мобільний додаток “Wallet” обчисливши суму усіх “rcptAmount” та додавши до неї суму вибраних бонусів ПЛ торгової точки відправляє APDU команду “PUT DATA” на термінал для використання вказаної кількості бонусів при виконанні платежу, як це показано на рис.3.18.

Wallet: APDU command “PUT DATA”
Data type: “Card Data” (0x66), len = (0x1D) “29”
Tag: “PAN 16” (0x5A10), “185DFE6ABBBF4171992F60CDB3F6E468”
Tag: “Amount, Other, Exp” (0x5F33), “3”
Tag: “Amount, Other” (0x9F03), “14.615”

80	CB	00	66	1D

5A	10	18	5D	FE	6A	BB	BF	41	71	99	2F	60	CD	B3	F6	E4	68

5F	33	03	9F	03	00	00	00	01	46	15	00

Terminal: APDU response
Data type: “Card Data” (0x66)
Tag: “Amount, Authorised, Exp” (0x5F32), “2”
Tag: “Amount, Authorised” (0x9F02), “109.92”
Tag: “Amount, Other, Exp” (0x5F33), “3”
Tag: “Amount, Other” (0x9F03), “0.073”
Rspcode: 0x9000

66	5F	32	02	9F	02	00	00	00	01	09	92

5F	33	03	9F	03	00	00	00	00	00	73	90	00

Рисунок 3.18 – APDU команда на використання бонусів ПЛ

В запиту в якості “PAN” вказується номер картки лояльності клієнта (“loyaltyUserId”), а сума бонусів для використання вказується параметром “Amount, Other”. Відповідь терміналу містить залишок суми для оплати фіатними коштами “Amount, Authorised” (різниця між сумою чеку та вартістю використаних бонусів в фіатній валюті) та суму бонусів, яка нараховується клієнту за здійснення платежу “Amount, Other”.

Початковий стан та результати виконання наведеного вище тестового прикладу виконання платіжної транзакції приведено в таблиці 3.4.

Таблиця 3.4 – Моніторинг виконання обміну бонусів клієнта

ПЛ	До обміну		Після обміну	
	ПЛ	Клієнт	ПЛ	Клієнт
“АЗС ОККО” Buy: 2940 Sell: 3060	1279340 CC	1.054 LTR	1320838 CC	14.615 LTR
“МАУ” Buy: 1960 Sell: 2040	3598728 CC	20 MIL	3559528 CC	0 MIL
“Сільпо” Buy: 0.98 Sell: 1.02	4677396 CC	2345 SLP	4675098 CC	0 SLP
“Епіцентр” Buy: 96 Sell: 104	5728430 CC	67.08 EPC	5728430 CC	67.08 EPC

Перевіримо коректність функціонування системи виконавши розрахунки. Нехай:

- N_LS – кількість зареєстрованих ПЛ;
- SEL_LS – індекс ПЛ торгової точки в множині програм лояльності;
- SC_LSпоч – множина початкових значень балансів рахунків в криптовалюті програм лояльності;
- SC_LSкін – множина значень балансів рахунків в криптовалюті програм лояльності після операцій обміну;
- R_BUY – множина поточних курсів продажу програм лояльності;
- R_SELL – множина поточних курсів купівлі програм лояльності;

- SB_CLпоч – множина початкових значень балансів в бонусах клієнта в кожній з ПЛ;
- SB_CLкін – множина значень балансів в бонусах клієнта в кожній з ПЛ після операції обміну;
- TB_CL – множина значень кількості бонусів які клієнт бажає обміняти.

Як видно з таблиці 3.4:

N_LS = 4, SEL_LS = 1,

SC_LSпоч = {1279340, 3598728, 4677396, 5728430},

R_BUY = {2940, 1960, 0.98, 96}, R_SELL = {3060, 2040, 1.02, 104},

SB_CLпоч = {1.054, 20, 2345, 67.08},

TB_CL = {0, 20, 2345, 0}

Елементи множини балансів програм лояльності після операції обміну розраховуються за формулою:

$$SC\_LS\text{кін}_i = \begin{cases} SC\_LS\text{поч}_i - R\_BUY_i \cdot TB\_CL_i \text{ для } i \neq SEL\_LS \\ SC\_LS\text{поч}_i + \sum_{j=1}^{N\_LS} R\_BUY_j \cdot TB\_CL_j \text{ для } i = SEL\_LS \end{cases} \quad (3.1)$$

Елементи множини балансів бонусних рахунків клієнта після операції обміну розраховуються за формулою:

$$SB\_CL\text{кін}_i = \begin{cases} SB\_CL\text{поч}_i - TB\_CL_i \text{ для } i \neq SEL\_LS \\ SB\_CL\text{поч}_i + \sum_{j=1}^{N\_LS} \frac{R\_BUY_j \cdot TB\_CL_j}{R\_SELL_i} \text{ для } i = SEL\_LS \end{cases} \quad (3.2)$$

Виконавши розрахунки за формулами (3.1) і (3.2), отримаємо наступні значення:

SC_LSкін = {1320838, 3559528, 4675098, 5728430},

SB_CLкін = {14.615, 0, 0, 67.08}

Отримані розрахункові дані повністю співпадають з даними моніторингу кінцевого стану системи приведеному в таблиці 3.4.

Висновок до розділу 3

Концепція гаманця клієнта дозволила організувати доступ користувача до окремих програм лояльності системи використовуючи єдиний профіль авторизації. Також це суттєво спрощує механізм оплати в торгових точках та процес обміну бонусів між програмами лояльності.

Оскільки для авторизації використовується протокол OAuth, в подальшому можливо підключати доступ до системи через стороні авторизаційні сервіси (Google, Facebook, тощо).

Відкритий REST API “Wallet Management System” дозволяє використовувати в якості клієнтських додатків стороннє програмне забезпечення (наприклад, існуючі мобільні додатки торгових мереж).

Використання стандарту ISO/IEC 7816 та технології NFC забезпечує взаємодію з платіжними терміналами торгівельних мереж згідно вимог міжнародних стандартів.

Механізм платіжної транзакції максимально наближений до механізму, який використовується Google Pay та в мобільному банкінгу. Отримані результати транзакції співпадають з результатами розрахункової моделі, що підтверджує коректну роботу системи в цілому.

РОЗДІЛ 4. ПРОГРАМИ ЛОЯЛЬНОСТІ ТА РОЗРАХУНКИ МІЖ НИМИ

4.1. Підключення програми лояльності

При розробці процедури підключення було враховано, що одними з основних цілей були прийняті: забезпечення максимальної довіри між учасниками (включно з організатором системи) та зняття обмежень на кількість програм лояльності. Для цього ключові дії виконуються власником нової ПЛ, а організатор системи тільки верифікує їх. Крім того, оскільки використовуються окремі канали блокчейн для транзакційного модулю та криптовалютні кроскурси обміну бонусів – підключення нової ПЛ не потребує додаткових дій від власників інших ПЛ.

Першим кроком, власник нової ПЛ реєструється як організація в центрі сертифікації ключів (ЦСК) та отримує наступні X.509 сертифікати з приватними ключами:

- сертифікат для роботи зі смарт-контрактами;
- сертифікати для кожного з власних вузлів, які власник бажає підключити до мережі блокчейн.

Організатор системи створює окремий канал для транзакційного модулю нової ПЛ, та конфігурує доступ до нього власних вузлів та вузлів власника ПЛ. Також в конфігурацію модулю забезпечення консенсусу вноситься інформація про створений канал. Вузли інших учасників не потребують модифікації конфігурації, оскільки вони довіряють сертифікатам даного ЦСК.

Власник ПЛ реєструється в системі (виклик методу "*lsCreate*" смарт-контракту "*LoyaltySystemContract*"). При цьому, для нього відкривається криптовалютний рахунок з нульовим балансом. Далі, для можливості обміну бонусів з іншими ПЛ, він задає криптовалютні курси продажу та купівлі для бонусів нової ПЛ (метод "*rateChange*" " смарт-контракту "*LoyaltySystemContract*"). Потім власник ПЛ, використовуючи шаблон, створює смарт-контракт транзакційного модулю (можливо з деякими

особливостями), завантажує його в новий канал та ініціює його (виклик методу “*init*”). Всі свої дії власник ПЛ виконує з використанням отриманого сертифікату для роботи зі смарт-контрактами. Останнім кроком власника ПЛ є розгортання на своїй території облікового модулю ПЛ (або як альтернатива – використання його, як сервісу платформи організатора системи). При цьому конфігурується дозвіл доступу до даного модулю з модулю керування гаманцями клієнтів та доступ даного модулю до блокчейн мережі за допомогою сертифікату для роботи зі смарт-контрактами отриманому власником ПЛ. Після завершення вищенаведених кроків ПЛ вважається створеною та сконфігурованою, але не активованою.

Активація ПЛ виконується організатором системи та складається з наступних кроків. Першим кроком організатор підключає ПЛ до тарифного плану розрахункового модулю (метод “*tarifSet*” смарт-контракту “*SettlementContract*”). Потім верифікує коректність алгоритму транзакційного модулю ПЛ (метод “*wmsAccept*” смарт-контракту “*TransExtrnContract*”). Останнім кроком, організатор системи підключає ПЛ до модулю керування гаманцями клієнтів (POST метод “ls” Administration API).

В мобільні гаманці клієнтів картки лояльності нової ПЛ будуть завантажені автоматично при першому зверненні до WMS (GET запит “wu” Wallet API). Також, після активації ПЛ, з’являється можливість виконання платіжних транзакцій через термінали торгової мережі власника програми лояльності.

4.2. Емісія криптовалюти

Оскільки в платіжних транзакціях в торговій точці, клієнти можуть використовувати бонуси ПЛ з інших торгівельних мереж (обмін бонусів) у власника даної мережі виникає необхідність отримання від інших учасників коштів за дані бонусні бали. При цьому виникають наступні проблеми:

- необхідність встановлення “справедливого” курсу обміну між різними ПЛ;
- захист від шахрайських дій учасника системи (неконтрольована емісія бонусів ПЛ);
- мотивація клієнта до активного використання ним бонусів з пріоритетом на власну торгову мережу;
- можливість розрахунків між учасниками міжнародної системи (з різних країн, з різними фіатними валютами);
- гарантії розрахунків учасників за використання їх бонусів в інших мережах.

Для їх вирішення, в розробленій системі, розрахунки ведуться через єдину криптовалюту системи. Її облік ведеться децентралізованою платіжною системою (без внутрішнього або зовнішнього адміністрування) в повністю автоматичному режимі за допомогою смарт-контрактів. При цьому облік криптовалютних рахунків ведеться модулем керування програмами лояльності, транзакції між рахунками виконуються за допомогою модуля операцій в криптовалюті, а розрахунок між учасниками здійснюється модулем розрахунків між програмами лояльності.

Власник ПЛ може отримати криптовалюту двома шляхами:

- надати можливість клієнту розрахуватися бонусами інших ПЛ;
- виконати емісію криптовалюти під гарантійні зобов’язання в фіатній валюті.

Система веде для кожного учасника два балансових рахунка в криптовалюті:

емісійний рахунок – поточний об’єм емітованої даним учасником криптовалюти;

поточний рахунок – загальний об’єм криптовалюти, доступної учаснику (сума власної емісії та коштів отриманих за погашення бонусів від інших ПЛ).

Цінісність бонусів ПЛ визначається їх обміним курсом до криптовалюти. Він встановлюється власником ПЛ (метод “rateChange” смарт-контракту “LoyaltySystemContract”). Однак шахрайські дії з ним виключені, оскільки він є ринковим та обмеженим поточним рахунком учасника:

- заниження курсу продажу зменшує використання клієнтом бонусів, а отже і їх привабливість (цінність);
- завищення курсу продажу зменшує об’єм їх максимальної кількості (емісії), оскільки неможливо продати більше ніж є на поточному рахунку коштів;
- заниження курсу купівлі підвищує привабливість розрахунків в даній торгівельній мережі, однак веде до зменшення отриманих коштів від інших учасників системи;
- завищення курсу купівлі зменшує привабливість торгової мережі у клієнтів.

Отже кожен власник, без суттєвого впливу інших ПЛ та організатора, вибудовує свої взаємовідносини з клієнтами:

- для підвищення лояльності клієнта до торгової мережі - підвищуючи курс продажу та занижуючи курс купівлі (при цьому знадобляться додаткові емісійні фіатні кошти);
- зменшення витрат на ПЛ – зниження курсу продажу та збільшення маржі між курсами (однак при цьому зменшуються об’єми платіжних транзакцій).

Для спрощення верифікації організатором системи гарантійних зобов’язань учасникам на виконанні ними емісії криптовалюта системи ведеться як стейблкойн (stablecoin) [15]. Якщо система розгортається в межах однієї країни, прив’язка виконується до фіатної валюти цієї країни, в інших випадках – долар США.

Емісія криптовалюти відбувається наступним чином. Власник ПЛ відправляє організатору системи гарантійні зобов’язання (або переказує

кошти на овердрафт рахунок) в об’ємі додаткової емісії. Після чого, він виконує створення емісії криптовалюти (метод “emisCreate” смарт-контракту “CryptoCurrencyContract”). Створена емісійна транзакція знаходиться в стані “очікує верифікації”, а випущений об’єм криптовалюті в заблокованому стані, без дозволу на використання.

Організатор системи з періодичністю перевіряє систему на наявність таких транзакцій (метод “emisGetWait”). Після їх появи, він перевіряє наявність гарантій, та у випадку їх достовірності, верифікує емісійну транзакцію (метод “emisAccept”). При цьому даний об’єм криптовалюти розблоковується та зараховується на емісійний рахунок власника ПЛ, а транзакція переходить в стан “успіх”. Поточний баланс рахунку учасника системи також збільшується на суму емісії і криптовалюта може використовуватися в платіжних транзакціях, або для розрахунків з іншими учасниками.

Якщо організатор системи відмовляє в достовірності гарантій, він виконує метод “emisReject” (з вказанням причини відмови). Емісійна транзакція переходить в стан “відмова” і даний об’єм криптовалюти анулюється.

До моменту верифікації емісійної транзакції, власник ПЛ, який її створив, може виконати скасування емісії (метод “emisCancel”). При цьому емісійна транзакція переходить в стан “скасовано” і даний об’єм криптовалюти анулюється.

Власники ПЛ можуть виконувати емісії криптовалюти в будь який час, та в будь яких об’ємах без погодження з іншими учасниками або організатором (але в рамках гарантійних зобов’язань). Це дозволяє їм гнучко налаштовувати роботу системи під свої потреби. В подальшому можливе розширення функціоналу системи виводом коштів з системи (погашенням емісії).

Організатор системи не має можливості виконувати емісію криптовалюти та адмініструвати її. Його функції подібні функціям нотаріусам: верифікація гарантійних зобов’язань та ведення їх реєстру.

4.3. Розрахунок між програмами лояльності

Розрахунки в системі складаються з розрахунку за обмін бонусів ПЛ та розрахунку комісійної винагороди за виконання транзакції.

Розрахунок за обмін бонусів ПЛ здійснюється одночасно з виконанням клієнтської транзакції. При цьому, з поточного рахунку власника ПЛ бонуси якої обмінюються, списується наступна сума в криптовалюті:

$$TC = TB\_CL_1 \cdot R\_BUY_1 \quad (4.1)$$

де *TC* – сума транзакції в криптовалюті;

TB_CL_1 – кількість бонусів ПЛ для обміну;

R_BUY_1 – курс продажу ПЛ, з якої клієнт обмінює бонуси.

Емісійний рахунок власника даної ПЛ при цьому залишається незмінним. Оскільки клієнтська транзакція складається з двох етапів, як це показано в розділі 3.2, сума спочатку зараховується на тимчасовий рахунок і тільки на етапі зарахування клієнтові бонусів, система виконує зарахування даної суми на поточний рахунок власника ПЛ торгової точки. При цьому, емісійний рахунок власника ПЛ торгової точки також залишається незмінним. Клієнт отримує можливість використати наступну кількість бонусів для оплати в торговій мережі:

$$TB\_CL_2 = TC/R\_SELL_2 \quad (4.2)$$

де TB_CL_2 – кількість бонусів ПЛ для платежу;

TC – сума транзакції в криптовалюті;

R_SELL_2 – курс купівлі ПЛ торгової точки.

Розрахунок комісійної винагороди здійснюється виконанням смарт-контракту розрахункового модулю. Для цього, організатор системи через backoffice систему, або через планувальник завдань виконує метод "*settlementCalc*" контракту "*SettlementContract*". Алгоритм розрахунку представлено на схемі ІАЛЦ.467200.007 Д4.

Спочатку обраховуються прогнозовані витрати власників ПЛ на комісію організатору системи. Для цього формується дві множини транзакцій даної ПЛ в яких поле СТАН_РОЗРАХ = 'Ні' (перша – транзакції

продажу бонусів, друга – транзакції купівлі бонусів). Потім обраховується сума витрат, згідно тарифного плану для даної ПЛ:

$$\text{ВИТР}_{\text{пл}_i} = \sum_{j=1}^{N_{tbuy_i}} (TC_{buy_j} \cdot prc_{buy_i} + val_{buy_i}) + \sum_{j=1}^{N_{tsell_i}} (TC_{sell_j} \cdot prc_{sell_i} + val_{sell_i}) \quad (4.3)$$

де $\text{ВИТР}_{\text{пл}_i}$ – сума комісії в криптовалюті за виконання транзакцій окремої ПЛ;

N_{tbuy_i} – розмірність множини транзакцій продажу окремої ПЛ;

TC_{buy_j} – сума транзакцій продажу в криптовалюті;

prc_{buy_i} – процент комісії за транзакцію продажу;

val_{buy_i} – фіксована величина комісії в криптовалюті за транзакцію продажу;

N_{tsell_i} – розмірність множини транзакцій купівлі окремої ПЛ;

TC_{sell_j} – сума транзакцій купівлі в криптовалюті;

prc_{sell_i} – процент комісії за транзакцію купівлі;

val_{sell_i} – фіксована величина комісії в криптовалюті за транзакцію купівлі.

Оскільки оплата витрат здійснюється в першу чергу за рахунок погашення криптовалюти, отриманої власниками ПЛ за транзакції купівлі від інших учасників, формується дві множини:

- ВИМОГпл – для ПЛ, у яких баланс поточного рахунку перевищує об’єм власної емісії криптовалюти;
- БОРГпл – для ПЛ, у яких баланс поточного рахунку менше об’єму власної емісії криптовалюти.

Вимоги на погашення криптовалюти в рахунок оплати комісії обраховуються за формулою:

$$\text{ВИМОГ}_{\text{пл}_i} = \min\left(\text{ВИТР}_{\text{пл}_i}, S_{tot_i} - S_{emis_i}\right) \quad (4.4)$$

де $\text{ВИМОГ}_{\text{пл}_i}$ – сума витрат на комісію окремої ПЛ, яка сплачується погашенням криптовалюти інших ПЛ;

$\text{ВИТР}_{\text{пл}_i}$ – сума комісії в криптовалюті за виконання транзакцій окремої ПЛ;

S_{tot_i} – баланс поточного рахунку ПЛ;

S_{emis_i} – баланс емісійного рахунку ПЛ.

Поточні боргові зобов'язання ПЛ розраховуються за формулою:

$$\text{БОРГ}_{\text{пл}_i} = S_{emis_i} - S_{tot_i} \quad (4.5)$$

де $\text{БОРГ}_{\text{пл}_i}$ – повна сума боргових зобов'язань ПЛ перед початком взаєморозрахунків;

S_{emis_i} – баланс емісійного рахунку ПЛ;

S_{tot_i} – баланс поточного рахунку ПЛ

Залишок суми комісії власник ПЛ сплачує за рахунок власних емісійних коштів:

$$\text{ЕМІС}_{\text{пл}_i} = \text{ВИТР}_{\text{пл}_i} - \text{ВИМОГ}_{\text{пл}_i} \quad (4.6)$$

де $\text{ЕМІС}_{\text{пл}_i}$ – сума витрат на комісію окремої ПЛ, яка сплачується власними емісійними коштами;

$\text{ВИТР}_{\text{пл}_i}$ – сума комісії в криптовалюті за виконання транзакцій окремої ПЛ;

$\text{ВИМОГ}_{\text{пл}_i}$ – сума витрат на комісію окремої ПЛ, яка сплачується погашенням криптовалюти інших ПЛ.

Оскільки ведення для поточного рахунку детального обліку ким була емітована окрема частина балансу суттєво збільшує об'єм реєстру та знижує швидкодію ми використали принцип пропорційної відповідальності боржників. Чим більший борг власника ПЛ, тим більшу частку загальної суми вимог на погашення він сплачує. Оплата боргу обраховується за формулою:

$$\text{ОПЛБОРГ}_{\text{пл}_i} = \left(\sum_{j=1}^{N_{\text{вимог}}} \text{ВИМОГ}_{\text{пл}_j}\right) \cdot \text{БОРГ}_{\text{пл}_i} / \sum_{j=1}^{N_{\text{борг}}} \text{БОРГ}_{\text{пл}_j} \quad (4.7)$$

де $\text{ОПЛБОРГ}_{\text{пл}_i}$ – частка вимог до погашення, яка сплачується окремим боржником;

$N_{\text{вимог}}$ – розмір множини ПЛ, які висунули вимоги на погашення;

$\text{ВИМОГ}_{\text{пл}_j}$ – сума витрат на комісію окремої ПЛ, яка сплачується погашенням криптовалюти інших ПЛ;

$\text{БОРГ}_{\text{пл}_i}, \text{БОРГ}_{\text{пл}_j}$ – повна сума боргових зобов'язань ПЛ перед початком взаєморозрахунків;

$N_{\text{вимог}}$ – розмір множини ПЛ, які мають борги.

Після проведення вказаних вище розрахунків, організатор системи виставляє учасникам системи рахунки на виконані роботи за звітний період на суму $\text{ВИТР}_{\text{пл}}$, однак до сплати вказується значення $\text{ЕМІС}_{\text{пл}}+\text{ОПЛБОРГ}_{\text{пл}}$.

Відповідно зменшується баланс емісійного рахунку:

$$S_{emis_i} = S_{emis_i} - (\text{ЕМІС}_{\text{пл}_i} + \text{ОПЛБОРГ}_{\text{пл}_i}) \quad (4.8)$$

де S_{emis_i} – баланс емісійного рахунку ПЛ;

$\text{ЕМІС}_{\text{пл}_i}$ – сума витрат на комісію окремої ПЛ, яка сплачується власними емісійними коштами;

$\text{ОПЛБОРГ}_{\text{пл}_i}$ – частка вимог до погашення, яка сплачується окремим боржником.

Нове значення балансу поточного рахунку обраховується за формулою:

$$S_{tot_i} = S_{tot_i} - \text{ВИМОГ}_{\text{пл}_i} - \text{ЕМІС}_{\text{пл}_i} + \text{ОПЛБОРГ}_{\text{пл}_i} \quad (4.9)$$

де S_{tot_i} – баланс поточного рахунку ПЛ;

$\text{ЕМІС}_{\text{пл}_i}$ – сума витрат на комісію окремої ПЛ, яка сплачується власними емісійними коштами;

$\text{ОПЛБОРГ}_{\text{пл}_i}$ – частка вимог до погашення, яка сплачується окремим боржником.

Для перевірки коректності роботи розрахункового модулю було виконано наступний тестовий приклад. Поточний стан гаманця клієнта та курси обміну ПЛ представлені у таблиці 4.1.

Таблиця 4.1 – Гаманець клієнта до виконання операцій

Програма лояльності	Баланс	Купівля	Продаж
okko	100.000	2940	3060
mau	100	1960	2040
silpo	100000	0.98	1.02
epicenter	10000.00	96	104

Баланси поточного та емісійного рахунків ПЛ перед початком тестування наведено у таблиці 4.2.

Таблиця 4.2 - Баланси рахунків ПЛ перед початком тестування

Програма лояльності	Поточний	Емісійний
okko	1000000	1000000
mau	1000000	1000000
silpo	1000000	1000000
epicenter	1000000	1000000

Тарифні плани комісійної винагороди для усіх ПЛ становлять:

- для транзакцій продажу: 2%+10
- для транзакцій купівлі: 1%+5

Клієнтом було виконано три операції обміну:

1. (10 mau + 1000 silpo + 100.00 epicenter) на okko
2. (5.000 okko + 5 mau) на silpo
3. (15.000 okko + 15 mau + 5000 silpo) на epicenter

Після чого, гаманець клієнта мав значення, приведені у таблиці 4.3, а баланси рахунків ПЛ у таблиці 4.4.

Таблиця 4.3 – Гаманець клієнта після виконання операцій

Програма лояльності	Баланс
okko	89.863
mau	70
silpo	118020
epicenter	10653.85

Таблиця 4.4 - Баланси рахунків ПЛ після виконання операцій

Програма лояльності	Поточний	Емісійний
okko	971380	1000000
mau	941200	1000000
silpo	1018620	1000000
epicenter	1068800	1000000

Для перевірки, виконаємо розрахунки транзакцій за формулами (4.1) – (4.3). Результат обчислення наведено у таблиці 4.5.

Таблиця 4.5 – Розрахунок транзакцій тестових операцій

Опер.	Транз.	ПЛ	TB_CL_1	TB_CL_2	ТС	$ВИТР_{пл}$
1	продаж	mau	10		19600	402
1	продаж	silpo	1000		980	30
1	продаж	epicenter	100.00		9600	202
1	купівля	okko		9.863	30180	307
2	продаж	okko	5.000		14700	304
2	продаж	mau	5		9800	206
2	купівля	silpo		24020	24500	250
3	продаж	okko	15.000		44100	892
3	продаж	mau	15		29400	598
3	продаж	silpo	5000		4900	108
3	купівля	epicenter		753.85	78400	789

Якщо баланси гаманця клієнта з таблиці 4.1 зменшити на значення TB_CL1 та збільшити на TB_CL2 отримаємо значення, які співпадають зі значеннями таблиці 4.3. При зменшенні поточних балансів ПЛ з таблиці 4.2 на значення ТС для транзакцій продажу та збільшенні на ТС для транзакцій купівлі, отримаємо поточні баланси, які збігаються зі значеннями з таблиці 4.4. Отже розроблена система виконує операції обміну вірно, у відповідності до розрахункової моделі.

Після тестового розрахунку звітного періоду отримуємо значення приведені у таблиці 4.6.

Таблиця 4.6 – Стан системи після розрахунку звітного періоду

ПЛ	Поточний	Емісійний	Нараховано	До сплати
okko	970328	998046	1503	1954
mau	940922	997866	1206	2134
silpo	1018232	1000000	388	0
epicenter	1067809	1000000	991	0

Для перевірки, виконаємо розрахунки за формулами (4.4) – (4.7). Результати обчислень приведені у таблиці 4.7.

Таблиця 4.7 – Результати обчислень взаєморозрахунків

ПЛ	ВИТР	ВИМОГ	БОРГ	ЕМІС	ОПЛБОРГ
okko	1503		28620	1503	451
mau	1206		58800	1206	928
silpo	388	388			
epicenter	991	991			

Обрахувавши за формулами (4.8) і (4.9) поточні та емісійні баланси ми отримуємо значення, ідентичні приведеним у таблиці 4.6. Отже розроблена система функціонує вірно у відповідності з розрахунковою моделлю.

Як видно з аналізу отриманих результатів, запропонована розрахункова модель, на відміну від типових рішень, дозволяє розраховуватися валютою, отриманою від купівлі бонусів у інших учасників системи (ПЛ “silpo” і “epicenter”). Це являється додатковим стимулом участі власників ПЛ в процесі обміну бонусів. Такі учасники як “okko” і “mau” хоча і повинні сплатити фіатними коштами за користування системою, однак як показано в [3] значно розширюють базу клієнтів ПЛ, а також підвищують лояльність існуючих користувачів.

4.4. Відмовостійкість системи

Оскільки при розробці системи були використанні розподілена мережа блокчейн та технології кластеризації, усі модулі працюють в режимі горячого резервування. Це дозволяє, в разі необхідності, підвищувати надійність роботи системи за допомогою розгортання додаткових вузлів.

Ймовірність відмови системи при виконанні платіжної транзакції з одним обміном бонусних балів та забезпеченням консенсусу трьома вузлами блокчейн обраховується за наступною формулою:

$$Q_{\text{пл.тр}} = \prod_{i=1}^{N_{wms}} Q_{wms_i} + \prod_{i=1}^{N_{db}} Q_{db_i} + \prod_{i=1}^{N_{abuy}} Q_{abuy_i} + 3\prod_{i=1}^{N_{tbuy}} Q_{tbuy_i} + \prod_{i=1}^{N_{asell}} Q_{asell_i} + 3\prod_{i=1}^{N_{tsell}} Q_{tsell_i} + 9\prod_{i=1}^{N_{pub}} Q_{pub_i} \quad (4.10)$$

де $Q_{\text{пл.тр}}$ - ймовірність відмови системи при транзакції обміну;

N_{wms} – розмір кластеру модуля керування гаманцями;

Q_{wms_i} - ймовірність відмови вузла кластеру модуля керування гаманцями;

N_{db} – розмір кластеру СУБД;

Q_{db_i} - ймовірність відмови вузла кластеру СУБД;

N_{abuy} – розмір кластеру облікового модулю ПЛ з якої списуються бонуси;

Q_{abuy_i} - ймовірність відмови вузла кластеру облікового модулю ПЛ з якої списуються бонуси;

N_{tbuy} – розмір каналу транзакційного модулю ПЛ з якої списуються бонуси;

Q_{tbuy_i} - ймовірність відмови вузла транзакційного модулю ПЛ з якої списуються бонуси;

N_{asell} – розмір кластеру облікового модулю ПЛ в якій нараховуються бонуси;

Q_{asell_i} - ймовірність відмови вузла кластеру облікового модулю ПЛ в якій нараховуються бонуси;

N_{tsell} – розмір каналу транзакційного модулю ПЛ в якій нараховуються бонуси;

Q_{tsell_i} - ймовірність відмови вузла транзакційного модулю ПЛ в якій нараховуються бонуси;

N_{tsell} – розмір публічного каналу блокчейн мережі;

Q_{tsell_i} - ймовірність відмови вузла публічного каналу блокчейн мережі.

Розрахуємо максимальну величину ймовірності відмови при розгортанні системи в IBM Cloud. Для цього використаємо мінімальну конфігурацію системи, яка складається з:

1 вузол модулю керування гаманцями;

1 вузол СУБД;

1 вузол облікового модулю ПЛ, з якою списуються бонуси;

1 вузол блокчейн мережі власника ПЛ, з якою списуються бонуси;

1 вузол облікового модулю ПЛ, в якій нараховуються бонуси;

1 вузол блокчейн мережі власника ПЛ, в якій нараховуються бонуси;

3 вузли блокчейн мережі організатора системи.

Для вузла IBM Cloud ймовірність відмови протягом року складає $5*10^{-4}$. Підставивши значення в формулу (4.10) отримаємо ймовірність відмови системи протягом року:

$$Q_{\text{пл.тр}} = 10^{-4} + 10^{-4} + 10^{-4} + 3 \cdot (10^{-4})^3 + 10^{-4} + 3 \cdot (10^{-4})^3 + 9 \cdot (10^{-4})^5 \approx 0.0004$$

Час, протягом якого система не являється працездатною визначається наступною формулою:

$$T_{\text{відм}} = Q_{\text{пл.тр}} \cdot 365 \cdot 24 \qquad (4.11)$$

де $T_{\text{відм}}$ – час непрацездатності системи протягом року, годин;

$Q_{\text{пл.тр}}$ - ймовірність відмови системи на протязі року.

Підставивши отримані значення в формулу (4.11) отримаємо:

$$T_{\text{відм}} = 0.004 \cdot 365 \cdot 24 \approx 3.5 \text{ години}$$

Отримане значення не перевищує вимоги технічного завдання, отже використання апаратного забезпечення IBM Cloud задовольняє вимоги до надійності технічного завдання навіть в мінімальній конфігурації кластерів.

Збільшення кількості вузлів в кластерах модулів системи, як видно з формули (4.10) дозволяє, при необхідності, зменшити ймовірність відмови, а отже підвищити рівень надійності системи. Крім того блокчейн мережа автоматично відновлює реєстр даних при його втраті (синхронізація з іншими вузлами розподіленої системи).

4.5. Аналіз продуктивності системи

Для аналізу продуктивності системи було розроблено тестовий запит від мобільного гаманця на обмін бонусних балів який було розміщено в кластері з 10 вузлів. Кожен вузол генерував 10000 запитів, але враховував тільки ті, на які приходила успішна відповідь впродовж 60 секунд. Модулі

системи були розгорнуті на однакових вузлах IBM Cloud з наступною специфікацією: ЦПУ 2.4 Ггц, ОЗП 4Гб, Linux Ubuntu 22.04 LTS. Результати виконання для різних конфігурацій приведено в таблиці 4.8.

Таблиця 4.8 – Результати тесту на продуктивність системи

Конфігурація	Запитів за хвилину
WMS: 1, СУБД: 1 Організатор: 3 блокчейн ПЛ1: 1 облік.модуль, 1 блокчейн ПЛ2: 1 облік.модуль, 1 блокчейн	1908
WMS: 2, СУБД: 2 Організатор: 3 блокчейн ПЛ1: 1 облік.модуль, 1 блокчейн ПЛ2: 1 облік.модуль, 1 блокчейн	2007
WMS: 1, СУБД: 1 Організатор: 9 блокчейн ПЛ1: 1 облік.модуль, 1 блокчейн ПЛ2: 1 облік.модуль, 1 блокчейн	4934
WMS: 2, СУБД: 2 Організатор: 9 блокчейн ПЛ1: 2 облік.модуль, 1 блокчейн ПЛ2: 2 облік.модуль, 1 блокчейн	5376

Як видно з таблиці, навіть мінімальна конфігурація системи забезпечує досить високу продуктивність роботи. Найбільш суттєвий вплив вносить виконання смарт-контрактів та підпис блоків блокчейну. Однак збільшення вузлів блокчейну лінійно пропорційно збільшує продуктивність системи. Це дозволяє стверджувати, що розроблена система здатна забезпечити продуктивність на рівні платіжних систем (не менше 5 млн транзакцій на добу) при наявності відповідних апаратних ресурсів.

Висновок до розділу 4

Використовуєма в розробленій системі розрахункова модель, на відміну від більшості існуючих, вводить можливість переносу оплати комісійних платежів на інших учасників, як винагороду за надання можливості використання бонусів їхніх ПЛ в торговій мережі даного власника. Це значно підвищує ефективність використання системи та являється стимулюючим механізмом до приймання бонусів сторонніх ПЛ.

Проведення операцій через криптовалюту дозволило значно спростити керування системою, особливо при використані її на міжнародному рівні.

Використання кроскурсів бонусів через криптовалюту та обмеження операцій поточним балансом дозволило розробити автоматично регульовану систему в якій усуваються шахрайські дії з неконтрольованими об’ємами бонусів ПЛ.

Технологія смарт-контрактів та мережа блокчейн дозволили уникнути адміністрування системи. Організатор системи виконує лише функції верифікатора, та не має можливості емісії криптовалюти, виконання або модифікації транзакцій, а також їх видалення з системи.

ВИСНОВОК

В дипломному проєкті розроблено програмне забезпечення розрахунково-транзакційної системи для коаліційних програм лояльності:

- модуль керування програмами лояльності;
- модуль операцій в криптовалюті;
- обліковий модуль програми лояльності;
- транзакційний модуль програми лояльності;
- модуль розрахунків між програмами лояльності;
- мобільний гаманець клієнта;
- модуль керування гаманцями клієнтів.

Використання при розробці блокчейн платформи з підтримкою смарт-контрактів та каналами для розмежування доступу дозволило створити систему із захистом від модифікації транзакційних даних та захистом від несанкціонованого доступу до клієнтських даних. Використання швидкодійного алгоритму забезпечення консенсусу BFT та механізмів кластеризації забезпечує необхідну, згідно технічного завдання, продуктивність системи та можливість її масштабування. Відповідність зовнішніх інтерфейсів специфікаціям ISO/IEC 14443 та ISO/IEC 7816 дозволяє інтегрувати розроблену систему з існуючими платіжними терміналами торгівельних мереж.

В першому розділі проаналізовані існуючі програми лояльності та визначені основні проблеми, з якими стикаються їх розробники. Оскільки основною проблемою коаліційної програми лояльності є недовіра між її учасниками, сучасні системи почали використовувати технології смарт-контрактів та ведення транзакційних реєстрів в ланцюгах блоків блокчейн мережі. Більшість рішень на базі смарт-контрактів ведуть єдиний реєстр транзакцій без розмежування доступу до нього і розміщують його тільки на вузлах організатора системи. В запропонованому мною рішенні, реєстри транзакцій ведуться в окремих каналах, що дозволяє використовувати для їх зберігання вузли блокчейн власників програм лояльності, гарантуючи при

цьому, що данні їхніх клієнтів будуть недоступними іншим учасникам системи. Також, розроблена система, на відміну від аналогів, замість обміну бонусів між програмами лояльності напряму, використовує проміжні криптовалютні транзакції. Це дозволяє уникнути обмеження на кількість ПЛ (кроскурс обміну до криптовалюти суттєво зменшує розмір таблиці курсів, на відміну від варіанта «усі до усіх») та забезпечує виконання зобов'язань учасників системи (гарантії в межах балансу криптовалютного рахунку).

На основі аналізу було прийнято рішення для розробки використати блокчейн IBM Blockchain Platform зі смарт-контрактами на мові Java та бібліотеку Spring для реалізації REST інтерфейсів. Було створено структурну схему системи, яка визначила принцип взаємодії функціональних модулів між собою. Діаграма класів смарт-контрактів та специфікація інтерфейсів за допомогою OpenAPI дозволили вести розробку одночасно зі створенням проєктної документації.

Була розроблена та перевірена наступна функціональність системи:

- підключення нової ПЛ з організацією окремого каналу блокчейну для її реєстру транзакцій;
- емісія криптовалюти власником ПЛ та її верифікація організатором системи;
- реєстрація клієнта в системі та відкриття для нього гаманця для карток програм лояльності;
- оплата клієнтом через платіжний термінал з обміном бонусів інших програм лояльності;
- розрахунок комісії за платіжні транзакції;
- взаєморозрахунок між власниками ПЛ.

Проєкт виконано в повному обсязі, а розрахунки та виконання тестових прикладів підтверджують працездатність системи та задовольняють вимогам технічного завдання.

СПИСОК ЛІТЕРАТУРИ

1. Kristen McCormick. How Much Does Google Ads Cost in 2022? – WordStream – 2022. - https://www.wordstream.com/blog/ws/2015/05/21/how-much-does-adwords-cost

2. Nicole Leinbach-Reyhle. Customer Loyalty In Today's Modern Retail World – Forbes – 2016. - https://www.forbes.com/sites/nicoleleinbachreyhle/2016/04/20/customer-loyalty-in-todays-modern-retail-world/?sh=7804a6a8513c

3. Steve Fromhart, Lincy Therattil. Making blockchain real for customer loyalty rewards programs. – Deloitte Center for Financial Services – 2015.

4. Roger Wattenhofer. Blockchain Science: Distributed Ledger Technology - Inverted Forest Publishing – 2019. – 289 Pages

5. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. - BigData Congress, Honolulu, HI, USA – 2017. - pp. 557–564.

6. Lin, I., Liao, T. A Survey of Blockchain Security Issues and Challenges. - IJ Network Security – 2017. – pp. 653-659.

7. Michael, J., Cohn, A., Butcher, J. BlockChain technology. - The Journal – 2018. – pp. 34-44.

8. Hileman, D., Rauchs, M. Global Blockchain Benchmarking Study. - Cambridge Centre for Alternative Finance. Cambridge, United Kingdom, - 2018.

9. Башир И. Блокчейн: архитектура, криптовалюты, инструменты разработки, смарт-контракты. -М.: ДМК Пресс, 2019. – 538 с.

10. D. Hardt, Ed. The OAuth 2.0 Authorization Framework - Internet Engineering Task Force, RFC-6749 – 2012. – 76 Pages

11. M. Jones, J. Bradley, N. Sakimura. JSON Web Token (JWT) - Internet Engineering Task Force, RFC-7519 – 2015. – 30 Pages

12. ISO/IEC 14443-4:2018 Cards and security devices for personal identification — Contactless proximity objects — Part 4: Transmission protocol– ISO/IEC – 2018. – 55 Pages

13. ISO/IEC 7816-4:2020 Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange – ISO/IEC – 2020. – 176 Pages

14. ISO/IEC 7816-6:2020 Identification cards — Integrated circuit cards — Part 6: Interindustry data elements for interchange – ISO/IEC – 2020. – 28 Pages

15. S. Brooks, A. Jurisevic, M. Spain, K, Warwick. A decentralised payment network and stablecoin. – Havven – 2018. - https://cryptorating.eu/whitepapers /Havven/havven_whitepaper.pdf

ДОДАТОК А

Назва системи: Розрахунково-транзакційна система програм лояльності клієнтів на основі технології смарт-контрактів

Тип документа: специфікація

Кількість аркушів: 1

2022 рік

Позначення	Найменування	Примітки
Документація		
УКР.ВМУРоЛУУ.КС0220_04Б 35-1	Admin.pdf – Інсталяція та адміністрування системи	
УКР.ВМУРоЛУУ.КС0220_04Б 35-2	WMSGuide.pdf – Керування гаманцями клієнтів	
УКР.ВМУРоЛУУ.КС0220_04Б 35-3	UserGuide.pdf – Користувача мобільного гаманця	
УКР.ВМУРоЛУУ.КС0220_04Б 35-4	LSGuide.pdf – Керування програмою лояльності	
УКР.ВМУРоЛУУ.КС0220_04Б 81-1	Zapis.pdf – Пояснювальна записка	
Компоненти		
УКР.ВМУРоЛУУ.КС0220_04Б 12-1	github.com/Doggy228/AntoxaWallet – Текст програми мобільного гаманця	
УКР.ВМУРоЛУУ.КС0220_04Б 13-1	AntoxaWallet.pdf – Опис програми мобільного гаманця	
УКР.ВМУРоЛУУ.КС0220_04Б 12-2	github.com/Doggy228/WMS – Текст програми модулю керування гаманцями клієнтів	
УКР.ВМУРоЛУУ.КС0220_04Б 13-2	WMSOpenAPI.pdf – Опис програми модулю керування гаманцями клієнтів	
УКР.ВМУРоЛУУ.КС0220_04Б 12-3	github.com/Doggy228/LoyaltySystem – Текст програми облікового модулю програми лояльності	
УКР.ВМУРоЛУУ.КС0220_04Б 13-3	LSOpenAPI.pdf – Опис програми облікового модулю програми лояльності	
УКР.ВМУРоЛУУ.КС0220_04Б 12-4	LoyaltySystemContract.java – Текст програми модулю керування програмами лояльності	
УКР.ВМУРоЛУУ.КС0220_04Б 13-4	LoyaltySystemContract.pdf – Опис програми модулю керування програмами лояльності	
УКР.ВМУРоЛУУ.КС0220_04Б 12-5	CryptoCurrencyContract.java – Текст програми модулю операцій в криптовалюті	
УКР.ВМУРоЛУУ.КС0220_04Б 13-5	CryptoCurrencyContract.pdf – Опис програми модулю модулю операцій в криптовалюті	
УКР.ВМУРоЛУУ.КС0220_04Б 12-6	SettlementContract.java – Текст програми розрахункового модулю	
УКР.ВМУРоЛУУ.КС0220_04Б 13-6	SettlementContract.pdf – Опис програми розрахункового модулю	
УКР.ВМУРоЛУУ.КС0220_04Б 12-7	TransExtrnContract.java – Текст програми транзакційного модулю	
УКР.ВМУРоЛУУ.КС0220_04Б 13-7	TransExtrnContract.pdf – Опис програми транзакційного модулю	

ДОДАТОК Б

Назва системи: Розрахунково-транзакційна система програм лояльності клієнтів на основі технології смарт-контрактів

Назва підсистеми: Модуль керування програмами лояльності

Тип документа: текст програми

Код документу: УКР.ВМУРоЛУУ.КС0220_04Б 12-4

Кількість аркушів: 3

2022 рік

```
package edu.doggy228.antoxasmart;
import org.hyperledger.fabric.contract.*;
import org.hyperledger.fabric.contract.annotation.*;
import java.util.*;
@Contract(name = "LoyaltySystemContract")
@Default
public class LoyaltySystemContract implements ContractInterface {
    private static final String wms = "wms";
    private static final String lsList = "lsList";
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void init(Context ctx, String wmsId) {
        if (ctx.getStub().getState(wms) != null) throw new RuntimeException("Contract already init.");
        ctx.getStub().putStringState(wms, new Wms(wmsId, ctx.getClientIdentity().getId()).toJSONString());
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Wms wmsGet(Context ctx) {
        return Wms.fromJSONString(ctx.getStub().getStringState(wms));
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LoyaltySystem lsCreate(Context ctx, String lsId) {
        if (ctx.getStub().getPrivateData(lsList, lsId) != null)
            throw new RuntimeException("Loyalty system already exists.");
        LoyaltySystem ls = new LoyaltySystem();
        ls.setId(lsId);
        ls.setKey(ctx.getClientIdentity().getId());
        ls.setBalanceEmis(0);
        ls.setBalanceTotal(0);
        ls.setBalanceLock(0);
        ctx.getStub().putPrivateData(lsList, lsId, ls.toJSONString());
        return ls;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LoyaltySystem[] lsGetAll(Context ctx) {
        List<LoyaltySystem> res = new ArrayList<>();
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataByRange(lsList,"A","{");
        for(KeyValue el: it){
            res.add(LoyaltySystem.fromJSONString(el.getStringValue()));
        }
        return res.toArray(new LoyaltySystem[0]);
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LoyaltySystem lsGet(Context ctx, String lsId) {
        return LoyaltySystem.fromJSONString(ctx.getStub().getPrivateDataUTF8(lsList, lsId));
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Rate rateChange(Context ctx, String lsId, int scale, String rateBuy, String rateSell) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(ls.getKey()!=ctx.getClientIdentity().getId()) throw new RuntimeException("Access denied.");
        Rate rate = new Rate(scale, rateBuy, rateSell);
        ls.setRate(rate);
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return rate;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Rate rateGet(Context ctx, String lsId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        return ls.getRate();
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LoyaltySystem balanceEmisChange(Context ctx, String lsId, long amount) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
       if(Wms.fromJSONString(ctx.getStub().getStringState(wms)).getKey()!=ctx.getClientIdentity().getId()) throw new
RuntimeException("Access denied.");
        ls.setBalanceEmis(ls.getBalanceEmis()+amount);
        ls.setBalanceTotal(ls.getBalanceTotal()+amount);
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return ls;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LoyaltySystem balancesChange(Context ctx, String lsId, long amountChangeEmis, long amountChangeTotal) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(Wms.fromJSONString(ctx.getStub().getStringState(wms)).getKey()!=ctx.getClientIdentity().getId()) throw
new RuntimeException("Access denied.");
        ls.setBalanceEmis(ls.getBalanceEmis()+amountChangeEmis);
        ls.setBalanceTotal(ls.getBalanceTotal()+amountChangeTotal);
```

```
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return ls;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LockAmount lockCreate(Context ctx, String lsId, String extrnId, long amount) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(ls.getKey()!=ctx.getClientIdentity().getId()) throw new RuntimeException("Access denied.");
        for(LockAmount el: ls.getLockList()){
            if(el.getExtrnId().equals(extrnId)) return el;
        }
        if(amount>0 && ls.getBalanceTotal()-ls.getBalanceLock()<amount) throw new RuntimeException("Not enough
funds.");
        LockAmount lockAmount = new LockAmount(ctx.getStub().getTxId(), extrnId, amount);
        ls.getLockList().add(lockAmount);
        if(amount>0) ls.setBalanceLock(ls.getBalanceLock()+amount);
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return lockAmount;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LockAmount lockExec(Context ctx, String lsId, String lockId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(ls.getKey()!=ctx.getClientIdentity().getId()) throw new RuntimeException("Access denied.");
        LockAmount lockAmount = null;
        for(LockAmount el: ls.getLockList()){
            if(el.getId().equals(lockId)){
                lockAmount = el;
                break;
            }
        }
        if(lockAmount==null) throw new RuntimeException("Lock amount not found.");
        ls.setBalanceTotal(ls.getBalanceTotal()-lockAmount.getAmount());
        if(lockAmount.getAmount()>0) ls.setBalanceLock(ls.getBalanceLock()-lockAmount.getAmount());
        ls.getLockList().remove(lockAmount);
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return lockAmount;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LockAmount lockCancel(Context ctx, String lsId, String lockId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(ls.getKey()!=ctx.getClientIdentity().getId()) throw new RuntimeException("Access denied.");
        LockAmount lockAmount = null;
        for(LockAmount el: ls.getLockList()){
            if(el.getId().equals(lockId)){
                lockAmount = el;
                break;
            }
        }
        if(lockAmount==null) throw new RuntimeException("Lock amount not found.");
        if(lockAmount.getAmount()>0) ls.setBalanceLock(ls.getBalanceLock()-lockAmount.getAmount());
        ls.getLockList().remove(lockAmount);
        ctx.getStub().putPrivateData(lsId, ls.getId(), ls.toJSONString());
        return lockAmount;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public LockAmount lockGet(Context ctx, String lsId, String lockId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if(ls==null) throw new RuntimeException("Loyalty systam not found.");
        if(ls.getKey()!=ctx.getClientIdentity().getId()) throw new RuntimeException("Access denied.");
        for(LockAmount el: ls.getLockList()){
            if(el.getId().equals(lockId)){
                return el;
            }
        }
        return null;
    }
}

import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.*;
import org.hyperledger.fabric.contract.annotation.*;
@DataType() @Getter @Setter @AllArgsConstructor
public class Wms {
    @Property() private String id;
    @Property() private String key;
    public String toJSONString() {
        try {
```

```
            ObjectMapper mapper = new ObjectMapper();
            return mapper.writeValueAsString(this);
        } catch (Throwable e) {throw new RuntimeException(e);}
    }
    public static Wms fromJSONString(String json) {
        if(json==null) return null;
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.readValue(json.getBytes(StandardCharsets.UTF_8), Wms.class);
        } catch (Throwable e) {throw new RuntimeException(e);}
    }
}


@DataType() @Getter @Setter
public class LoyaltySystem {
    @Property() private String id;
    @Property() private String key;
    @Property() private long balanceEmis;
    @Property() private long balanceTotal;
    @Property() private long balanceLock;
    @Property() private Rate rate;
    @Property() private List<LockAmount> lockList;
…
}

@DataType() @Getter @Setter @AllArgsConstructor
public class Rate {
    @Property() private int scale;
    @Property() private String buy;
    @Property() private String sell;
…
}

@DataType() @Getter @Setter @AllArgsConstructor
public class LockAmount {
    @Property() private String id;
    @Property() private String extrnId;
    @Property() private long amount;
…
}
```

ДОДАТОК В

Назва системи: Розрахунково-транзакційна система програм лояльності клієнтів на основі технології смарт-контрактів

Назва підсистеми: Модуль операцій в криптовалюті

Тип документа: текст програми

Код документу: УКР.ВМУРоЛУУ.КС0220_04Б 12-5

Кількість аркушів: 5

2022 рік

```
package edu.doggy228.antoxasmart;
import org.hyperledger.fabric.contract.*;
import org.hyperledger.fabric.contract.annotation.*;
import org.hyperledger.fabric.shim.Chaincode;
import org.hyperledger.fabric.shim.ledger.*;
import java.math.*;
import java.time.ZoneId;
import java.util.*;
@Contract(name = "CryptoCurrencyContract")
@Default
public class CryptoCurrencyContract implements ContractInterface {
    private static final String wms = "wms";
    private static final String transEmisList = "transEmisList";
    private static final String transSettlementList = "transSettlementList";
    private static final String transBuyList = "transBuyList";
    private static final String transSellList = "transSellList";
    private Wms wmsGet(Context ctx) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "wmsGet");
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        Wms wms = Wms.fromJSONString(rsp.getStringPayload());
        if (wms == null) throw new RuntimeException("Wms not found");
        return wms;
    }
    private Wms wmsCheck(Context ctx) {
        Wms wms = wmsGet(ctx);
        if (!wms.getKey().equals(ctx.getClientIdentity().getId()))
            throw new RuntimeException("Wms incorrect");
        return wms;
    }
    private LoyaltySystem lsGet(Context ctx, String lsId) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lsGet",
lsId);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        LoyaltySystem ls = LoyaltySystem.fromJSONString(rsp.getStringPayload());
        if (ls == null) throw new RuntimeException("Loyalty system not found");
        return ls;
    }
    private LoyaltySystem lsCheck(Context ctx, String lsId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if (!ls.getKey().equals(ctx.getClientIdentity().getId()))
            throw new RuntimeException("Loyalty system incorrect");
        return ls;
    }
    private TransEmis loEmisGet(Context ctx, String transId) {
        TransEmis trans = TransEmis.fromJSONString(ctx.getStub().getPrivateDataUTF8(transEmisList, transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }
    private TransSettlement loSettlementGet(Context ctx, String transId) {
        TransSettlement trans = TransSettlement.fromJSONString(ctx.getStub().getPrivateDataUTF8(transSettlementList,
transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }
    private TransBuy loBuyGet(Context ctx, String transId) {
        TransBuy trans = TransBuy.fromJSONString(ctx.getStub().getPrivateDataUTF8(transBuyList, transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }
    private TransSell loSellGet(Context ctx, String transId) {
        TransSell trans = TransSell.fromJSONString(ctx.getStub().getPrivateDataUTF8(transSellList, transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void init(Context ctx, String wmsId) {
        Wms wms = wmsCheck(ctx);
        if (!wms.getId().equals(wmsId)) throw new RuntimeException("Wms incorrect.");
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis emisCreate(Context ctx, String lsId, long amount) {
        LoyaltySystem ls = lsCheck(ctx, lsId);
        TransEmis trans = new TransEmis();
        trans.setId(ctx.getStub().getTxId());
        trans.setLsId(lsId);
        trans.setAmount(amount);
        trans.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setState(TransEmis.STATE_WAITACCEPT);
        ctx.getStub().putPrivateData(transEmisList, trans.getId(), trans.toJSONString());
```

```
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis emisAccept(Context ctx, String transId) {
        wmsCheck(ctx);
        TransEmis trans = loEmisGet(ctx, transId);
        if (trans.getState() != TransEmis.STATE_WAITACCEPT) throw new RuntimeException("Bad status transaction.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract",
"balanceEmisChange", "" + trans.getAmount());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        trans.setState(TransEmis.STATE_OK);
        trans.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        ctx.getStub().putPrivateData(transEmisList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis emisReject(Context ctx, String transId, String stateMsg) {
        wmsCheck(ctx);
        TransEmis trans = loEmisGet(ctx, transId);
        if (trans.getState() != TransEmis.STATE_WAITACCEPT) throw new RuntimeException("Bad status transaction.");
        trans.setState(TransEmis.STATE_REJECT);
        trans.setStateMsg(stateMsg);
        trans.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        ctx.getStub().putPrivateData(transEmisList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis emisCancel(Context ctx, String transId, String stateMsg) {
        TransEmis trans = loEmisGet(ctx, transId);
        lsCheck(ctx, trans.getLsId());
        if (trans.getState() != TransEmis.STATE_WAITACCEPT) throw new RuntimeException("Bad status transaction.");
        trans.setState(TransEmis.STATE_CANCEL);
        trans.setStateMsg(stateMsg);
        trans.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        ctx.getStub().putPrivateData(transEmisList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis[] emisGetWait(Context ctx) {
        wmsCheck(ctx);
        List<TransEmis> res = new ArrayList<>();
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataQueryResult(transEmisList,
"{\"selector\":{\"state\":" + TransEmis.STATE_WAITACCEPT + "}}");
        for (KeyValue el : it) {
            res.add(TransEmis.fromJSONString(el.getStringValue()));
        }
        return res.toArray(new TransEmis[0]);
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis emisGetOne(Context ctx, String transId) {
        TransEmis trans = loEmisGet(ctx, transId);
        LoyaltySystem ls = lsGet(ctx, trans.getLsId());
        if (!ls.getKey().equals(ctx.getClientIdentity().getId())) {
            Wms wms = wmsGet(ctx);
            if (!wms.getKey().equals(ctx.getClientIdentity().getId())) throw new RuntimeException("Access denied.");
        }
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransEmis[] emisGetPeriod(Context ctx, String lsId, String dstart, String dend) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if (!ls.getKey().equals(ctx.getClientIdentity().getId())) {
            Wms wms = wmsGet(ctx);
            if (!wms.getKey().equals(ctx.getClientIdentity().getId())) throw new RuntimeException("Access denied.");
        }
        List<TransEmis> res = new ArrayList<>();
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataQueryResult(transEmisList,
"{\"selector\":{\"$and\":[{\"lsId\":\"" + lsId + "\"}," +
                "{\"dtCreate\":{\"$gte\":\"" + dstart + "\"}},{\"dtCreate\":{\"$lt\":\"" + dend + "\"}}]}}");
        for (KeyValue el : it) {
            res.add(TransEmis.fromJSONString(el.getStringValue()));
        }
        return res.toArray(new TransEmis[0]);
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSettlement settlementCreate(Context ctx, TransSettlement trans) {
        wmsCheck(ctx);
        trans.setId(ctx.getStub().getTxId());
        for (TransSettlementLs el : trans.getTransSettlementLsList()) {
```

```
            el.setId(trans.getId() + "-" + el.getLsId());
            Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract",
"balancesChange", el.getLsId(),
                    "-"+(el.getPayLoan()+el.getPayEmis()),""+(el.getPayLoan()-el.getPayEmis()));
            if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        }
        trans.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        ctx.getStub().putPrivateData(transSettlementList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSettlement settlementGetOne(Context ctx, String transId) {
        wmsCheck(ctx);
        TransSettlement trans = loSettlementGet(ctx, transId);
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public long settlementPrepareByLs(Context ctx, String lsId, String prcBuy, long valBuy, String prcSell, long
valSell) {
        wmsCheck(ctx);
        LoyaltySystem ls = lsGet(ctx, lsId);
        long amountTotal = 0;
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataQueryResult(transBuyList,
"{\"selector\":{\"lsId\":\"" + lsId + "\"," +
                "\"stateSettlement\":" + TransBuy.STATE_SETTLEMENT_NONE + "\"}}");
        for (KeyValue el : it) {
            TransBuy trans = TransBuy.fromJSONString(el.getStringValue());
            amountTotal = amountTotal+new BigDecimal(prcBuy).multiply(new
BigDecimal(trans.getCcAmount())).setScale(0, RoundingMode.HALF_UP).longValue()+valBuy;
            trans.setStateSettlement(TransBuy.STATE_SETTLEMENT_OK);
            ctx.getStub().putPrivateData(transBuyList, trans.getId(), trans.toJSONString());
        }
        it = ctx.getStub().getPrivateDataQueryResult(transSellList, "{\"selector\":{\"lsId\":\"" + lsId + "\"," +
                "\"stateSettlement\":" + TransSell.STATE_SETTLEMENT_NONE + "\"}}");
        for (KeyValue el : it) {
            TransSell trans = TransSell.fromJSONString(el.getStringValue());
            amountTotal = amountTotal+new BigDecimal(prcSell).multiply(new
BigDecimal(trans.getCcAmount())).setScale(0, RoundingMode.HALF_UP).longValue()+valSell;
            trans.setStateSettlement(TransSell.STATE_SETTLEMENT_OK);
            ctx.getStub().putPrivateData(transSellList, trans.getId(), trans.toJSONString());
        }
        return amountTotal;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSettlement[] settlementGetPeriod(Context ctx, String dstart, String dend) {
        wmsCheck(ctx);
        List<TransSettlement> res = new ArrayList<>();
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataQueryResult(transSettlementList,
"{\"selector\":{\"$and\":[" +
                "{\"dtCreate\":{\"$gte\":\"" + dstart + "\"}},{\"dtCreate\":{\"$lt\":\"" + dend + "\"}}]}}");
        for (KeyValue el : it) {
            res.add(TransSettlement.fromJSONString(el.getStringValue()));
        }
        return res.toArray(new TransSettlement[0]);
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransBuy buyExec(Context ctx, String lsId, long ccAmount, String extrnId, String bonusAmount) {
        LoyaltySystem ls = lsCheck(ctx, lsId);
        if (ls.getRate() == null || ls.getRate().getBuy() == null) throw new RuntimeException("Rate not defined.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lockCreate",
lsId, extrnId, "" + ccAmount);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        LockAmount lockAmount = LockAmount.fromJSONString(rsp.getStringPayload());
        TransBuy trans = new TransBuy();
        trans.setId(ctx.getStub().getTxId());
        trans.setLsId(lsId);
        trans.setBonusAmount(bonusAmount);
        trans.setBonusScale(ls.getRate().getScale());
        trans.setCcAmount(ccAmount);
        trans.setExtrnId(extrnId);
        trans.setLockId(lockAmount.getId());
        trans.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setState(TransBuy.STATE_WAITSELL);
        trans.setStateSettlement(TransBuy.STATE_SETTLEMENT_NONE);
        ctx.getStub().putPrivateData(transBuyList, trans.getId(), trans.toJSONString());
        rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lockExec", lsId,
trans.getLockId());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        return trans;
```

```
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransBuy buyRefund(Context ctx, String transId, String stateMsg) {
        wmsCheck(ctx);
        TransBuy trans = loBuyGet(ctx, transId);
        if (trans.getState() != TransBuy.STATE_WAITSELL && trans.getState() != TransBuy.STATE_OK)
            throw new RuntimeException("Not refund.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract",
"balancesChange", trans.getLsId(), "0", "" + trans.getCcAmount());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        trans.setDtRefund(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setState(TransBuy.STATE_REFUND);
        trans.setStateMsg(stateMsg);
        ctx.getStub().putPrivateData(transBuyList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransBuy buyGetOne(Context ctx, String transId) {
        TransBuy trans = loBuyGet(ctx, transId);
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSell sellExec(Context ctx, String lsId, String transBuyId, String extrnId, String bonusAmount) {
        LoyaltySystem ls = lsCheck(ctx, lsId);
        if (ls.getRate() == null || ls.getRate().getSell() == null) throw new RuntimeException("Rate not defined.");
        TransBuy transBuy = loBuyGet(ctx, transBuyId);
        if (transBuy.getState() != TransBuy.STATE_WAITSELL) throw new RuntimeException("Trans buy not found.");
        BigDecimal bonusAmountCalc = new BigDecimal(transBuy.getCcAmount()).divide(new
BigDecimal(ls.getRate().getSell()), ls.getRate().getScale(), RoundingMode.HALF_UP);
        if (!bonusAmountCalc.toPlainString().equals(bonusAmount)) throw new RuntimeException("Bad bonus amount.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lockCreate",
lsId, extrnId, "-" + transBuy.getCcAmount());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        LockAmount lockAmount = LockAmount.fromJSONString(rsp.getStringPayload());
        TransSell trans = new TransSell();
        trans.setId(ctx.getStub().getTxId());
        trans.setLsId(lsId);
        trans.setBonusAmount(bonusAmount);
        trans.setBonusScale(ls.getRate().getScale());
        trans.setCcAmount(transBuy.getCcAmount());
        trans.setExtrnId(extrnId);
        trans.setLockId(lockAmount.getId());
        trans.setTransBuyId(transBuy.getId());
        trans.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setState(TransSell.STATE_OK);
        trans.setStateSettlement(TransSell.STATE_SETTLEMENT_NONE);
        ctx.getStub().putPrivateData(transSellList, trans.getId(), trans.toJSONString());
        transBuy.setState(TransBuy.STATE_OK);
        transBuy.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        ctx.getStub().putPrivateData(transBuyList, transBuy.getId(), transBuy.toJSONString());
        rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lockExec", lsId,
trans.getLockId());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSell sellRefund(Context ctx, String transId, String stateMsg) {
        wmsCheck(ctx);
        TransSell trans = loSellGet(ctx, transId);
        if (trans.getState() != TransBuy.STATE_OK) throw new RuntimeException("Not refund.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract",
"balancesChange", trans.getLsId(), "0", "-" + trans.getCcAmount());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        trans.setDtRefund(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        trans.setState(TransBuy.STATE_REFUND);
        trans.setStateMsg(stateMsg);
        ctx.getStub().putPrivateData(transBuyList, trans.getId(), trans.toJSONString());
        return trans;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSell sellGetOne(Context ctx, String transId) {
        TransSell trans = loSellGet(ctx, transId);
        return trans;
    }
}

@DataType() @Getter @Setter
public class TransEmis {
```

```
    public static final int STATE_WAITACCEPT = 1;
    public static final int STATE_OK = 2;
    public static final int STATE_REJECT = 3;
    public static final int STATE_CANCEL = 4;
    @Property() private String id;
    @Property() private String lsId;
    @Property() private long amount;
    @Property() private ZonedDateTime dtCreate;
    @Property() private ZonedDateTime dtExec;
    @Property() private int state;
    @Property() private String stateMsg;
…
}
@DataType() @Getter @Setter
public class TransSettlement {
    public static final int STATE_CALC = 1;
    public static final int STATE_OK = 2;
    @Property() private String id;
    @Property() private long amount;
    @Property() private ZonedDateTime dtCreate;
    @Property() private ZonedDateTime dtExec;
    @Property() private List<TransSettlementLs> transSettlementLsList;
…
}
@DataType() @Getter @Setter
public class TransSettlementLs {
    @Property() private String id;
    @Property() private String lsId;
    @Property() private long payEmis;
    @Property() private long payLoan;
    @Property() private long amountPay;
    @Property() private long amountTotal;
…
}
@DataType() @Getter @Setter
public class TransBuy {
    public static final int STATE_WAITSELL = 1;
    public static final int STATE_OK = 2;
    public static final int STATE_REFUND = 3;
    public static final int STATE_SETTLEMENT_NONE = 0;
    public static final int STATE_SETTLEMENT_OK = 1;
    @Property() private String id;
    @Property() private String lsId;
    @Property() private String bonusAmount;
    @Property() private int bonusScale;
    @Property() private long ccAmount;
    @Property() private String extrnId;
    @Property() private String lockId;
    @Property() private ZonedDateTime dtCreate;
    @Property() private ZonedDateTime dtExec;
    @Property() private ZonedDateTime dtRefund;
    @Property() private int state;
    @Property() private String stateMsg;
    @Property() private int stateSettlement;
…
}
@DataType() @Getter @Setter
public class TransSell {
    public static final int STATE_OK = 2;
    public static final int STATE_REFUND = 3;
    public static final int STATE_SETTLEMENT_NONE = 0;
    public static final int STATE_SETTLEMENT_OK = 1;

    @Property() private String id;
    @Property() private String lsId;
    @Property() private String bonusAmount;
    @Property() private int bonusScale;
    @Property() private long ccAmount;
    @Property() private String extrnId;
    @Property() private String lockId;
    @Property() private ZonedDateTime dtCreate;
    @Property() private ZonedDateTime dtExec;
    @Property() private ZonedDateTime dtRefund;
    @Property() private int state;
    @Property() private String stateMsg;
    @Property() private String transBuyId;
    @Property() private int stateSettlement;
…
}
```

ДОДАТОК Г

Назва системи: Розрахунково-транзакційна система програм лояльності клієнтів на основі технології смарт-контрактів

Назва підсистеми: Розрахунковий модуль

Тип документа: текст програми

Код документу: УКР.ВМУРоЛУУ.КС0220_04Б 12-6

Кількість аркушів: 2

2022 рік

```
package edu.doggy228.antoxasmart;
import org.hyperledger.fabric.contract.*;
import org.hyperledger.fabric.contract.annotation.*;
import org.hyperledger.fabric.shim.Chaincode;
import org.hyperledger.fabric.shim.ledger.*;
@Contract(name = "SettlementContract")
@Default
public class SettlementContract implements ContractInterface {
    private static final String wms = "wms";
    private static final String tarifList = "tarifList";
    private Wms wmsGet(Context ctx) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "wmsGet");
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        Wms wms = Wms.fromJSONString(rsp.getStringPayload());
        if (wms == null) throw new RuntimeException("Wms not found");
        return wms;
    }
    private Wms wmsCheck(Context ctx) {
        Wms wms = wmsGet(ctx);
        if (!wms.getKey().equals(ctx.getClientIdentity().getId()))
            throw new RuntimeException("Wms incorrect");
        return wms;
    }
    private LoyaltySystem lsGet(Context ctx, String lsId) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lsGet",
lsId);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        LoyaltySystem ls = LoyaltySystem.fromJSONString(rsp.getStringPayload());
        if (ls == null) throw new RuntimeException("Loyalty system not found");
        return ls;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void init(Context ctx, String wmsId) {
        Wms wms = wmsCheck(ctx);
        if (!wms.getId().equals(wmsId)) throw new RuntimeException("Wms incorrect.");
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Tarif tarifSet(Context ctx, String lsId, String prcBuy, long valBuy, String prcSell, long valSell) {
        wmsCheck(ctx);
        Tarif tarif = new Tarif(lsId, prcBuy, valBuy, prcSell, valSell);
        ctx.getStub().putPrivateData(tarifList, tarif.getLsId(), tarif.toJSONString());
        return tarif;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Tarif tarifGetOne(Context ctx, String lsId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if (!ls.getKey().equals(ctx.getClientIdentity().getId())) {
            wmsCheck(ctx);
        }
        return Tarif.fromJSONString(ctx.getStub().getPrivateDataUTF8(tarifList, lsId));
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public Tarif[] tarifGetAll(Context ctx) {
        wmsCheck(ctx);
        List<Tarif> res = new ArrayList<>();
        QueryResultsIterator<KeyValue> it = ctx.getStub().getPrivateDataQueryResult(tarifList, "{\"selector\":{}}");
        for (KeyValue el : it) {
            res.add(Tarif.fromJSONString(el.getStringValue()));
        }
        return res.toArray(new Tarif[0]);
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransSettlement settlementCalc(Context ctx) {
        wmsCheck(ctx);
        Map<String, CalcLs> calcLsMap = new HashMap<>();
        long total = 0;
        long totalRefund = 0;
        long totalLoan = 0;
        TransSettlement trans = new TransSettlement();
        trans.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        Tarif[] tarifs = tarifGetAll(ctx);
        for (Tarif tarif : tarifs) {
            CalcLs calcLs = new CalcLs();
            calcLs.lsId = tarif.getLsId();
            calcLsMap.put(calcLs.lsId, calcLs);
            LoyaltySystem ls = lsGet(ctx, tarif.getLsId());
            calcLs.balanceEmis = ls.getBalanceEmis();
            calcLs.balanceTotal = ls.getBalanceTotal();
```

```
            Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract",
"settlementPrepareByLs",
                    tarif.getLsId(), tarif.getPrcBuy(), "" + tarif.getValBuy(), tarif.getPrcSell(), "" +
tarif.getValSell());
            if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
            calcLs.amountTotal = Long.parseLong(rsp.getStringPayload());
            long amountRefund = ls.getBalanceTotal() - ls.getBalanceEmis();
            if (amountRefund > 0) {
                calcLs.payRefund = Math.min(amountRefund, calcLs.amountTotal);
                totalRefund += calcLs.payRefund;
            } else {
                calcLs.balanceLoan = ls.getBalanceEmis() - ls.getBalanceTotal();
                totalLoan += calcLs.balanceLoan;
            }
            calcLs.payEmis = calcLs.amountTotal - calcLs.payRefund;
            total += calcLs.amountTotal;
            TransSettlementLs transLs = new TransSettlementLs();
            transLs.setLsId(calcLs.lsId);
            transLs.setAmountTotal(calcLs.amountTotal);
            transLs.setPayEmis(calcLs.payEmis);
            trans.getTransSettlementLsList().add(transLs);
        }
        trans.setAmount(total);
        for(TransSettlementLs transLs: trans.getTransSettlementLsList()){
            CalcLs calcLs = calcLsMap.get(transLs.getLsId());
            if(totalRefund>0 && calcLs.balanceLoan>0){
                calcLs.payLoan =
(BigInteger.valueOf(calcLs.balanceLoan).multiply(BigInteger.valueOf(totalRefund)).divide(BigInteger.valueOf(totalLoa
n)).longValue());
            }
            transLs.setPayLoan(calcLs.payLoan);
        }
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract",
"settlementCreate", trans.toJSONString());
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        trans = TransSettlement.fromJSONString(rsp.getStringPayload());
        return trans;
    }
}

@DataType() @Getter @Setter @AllArgsConstructor
public class Tarif {
    @Property() private String lsId;
    @Property() private String prcBuy;
    @Property() private long valBuy;
    @Property() private String prcSell;
    @Property() private long valSell;

    public String toJSONString(){
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.writeValueAsString(this);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }

    public static Tarif fromJSONString(String json) {
        if (json == null) return null;
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.readValue(json.getBytes(StandardCharsets.UTF_8), Tarif.class);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }
}
```

ДОДАТОК Д

Назва системи: Розрахунково-транзакційна система програм лояльності клієнтів на основі технології смарт-контрактів

Назва підсистеми: Транзакційний модуль

Тип документа: текст програми

Код документу: УКР.ВМУРоЛУУ.КС0220_04Б 12-7

Кількість аркушів: 4

2022 рік

```
package edu.doggy228.antoxasmart;
import org.hyperledger.fabric.contract.*;
import org.hyperledger.fabric.contract.annotation.*;
import org.hyperledger.fabric.shim.Chaincode;
import org.hyperledger.fabric.shim.ledger.*;
import org.hyperledger.fabric.shim.ledger.QueryResultsIterator;
@Contract(name = "TransExtrnContract")
@Default
public class TransExtrnContract implements ContractInterface {
    private static final String wms = "wms";
    private static final String lsExtrn = "lsExtrn";
    private static final String transExtrnBuyList = "transExtrnBuyList";
    private static final String transExtrnSellList = "transExtrnSellList";
    private Wms wmsGet(Context ctx) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "wmsGet");
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        Wms wms = Wms.fromJSONString(rsp.getStringPayload());
        if (wms == null) throw new RuntimeException("Wms not found");
        return wms;
    }
    private Wms wmsCheck(Context ctx) {
        Wms wms = wmsGet(ctx);
        if (!wms.getKey().equals(ctx.getClientIdentity().getId())) throw new RuntimeException("Wms incorrect");
        return wms;
    }
    private LoyaltySystem lsGet(Context ctx, String lsId) {
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("LoyaltySystemContract", "lsGet",
lsId);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        LoyaltySystem ls = LoyaltySystem.fromJSONString(rsp.getStringPayload());
        if (ls == null) throw new RuntimeException("Loyalty system not found");
        return ls;
    }
    private LoyaltySystem lsCheck(Context ctx, String lsId) {
        LoyaltySystem ls = lsGet(ctx, lsId);
        if (!ls.getKey().equals(ctx.getClientIdentity().getId()))
            throw new RuntimeException("Loyalty system incorrect");
        return ls;
    }
    private LoyaltySystemExtrn loLsExtrnGet(Context ctx) {
        LoyaltySystemExtrn ls = LoyaltySystemExtrn.fromJSONString(ctx.getStub().getStringState(lsExtrn));
        if (ls == null) throw new RuntimeException("Loyalty system not found");
        return ls;
    }
    private LoyaltySystemExtrn loLsExtrnCheck(Context ctx) {
        LoyaltySystemExtrn lse = loLsExtrnGet(ctx);
        if (!lse.getKey().equals(ctx.getClientIdentity().getId()))
            throw new RuntimeException("Loyalty system incorrect");
        if (lse.getState() != LoyaltySystemExtrn.STATE_OK) throw new RuntimeException("Loyalty system not accept.");
        return lse;
    }
    private TransExtrnBuy loBuyExtrnGet(Context ctx, String transId) {
        TransExtrnBuy trans = TransExtrnBuy.fromJSONString(ctx.getStub().getPrivateDataUTF8(transExtrnBuyList,
transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }
    private TransExtrnSell loSellExtrnGet(Context ctx, String transId) {
        TransExtrnSell trans = TransExtrnSell.fromJSONString(ctx.getStub().getPrivateDataUTF8(transExtrnSellList,
transId));
        if (trans == null) throw new RuntimeException("Transaction not found");
        return trans;
    }

    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void init(Context ctx, String lsId) {
        LoyaltySystem ls = lsCheck(ctx, lsId);
        LoyaltySystemExtrn lse = new LoyaltySystemExtrn();
        lse.setId(lsId);
        lse.setKey(ctx.getClientIdentity().getId());
        lse.setState(LoyaltySystemExtrn.STATE_WAITACCEPT);
        ctx.getStub().putStringState(lsExtrn, lse.toJSONString());
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void wmsAccept(Context ctx, String wmsId) {
        wmsCheck(ctx);
        LoyaltySystemExtrn lse = loLsExtrnGet(ctx);
        if (lse.getState() != LoyaltySystemExtrn.STATE_WAITACCEPT) throw new RuntimeException("State not wait
accept.");
```

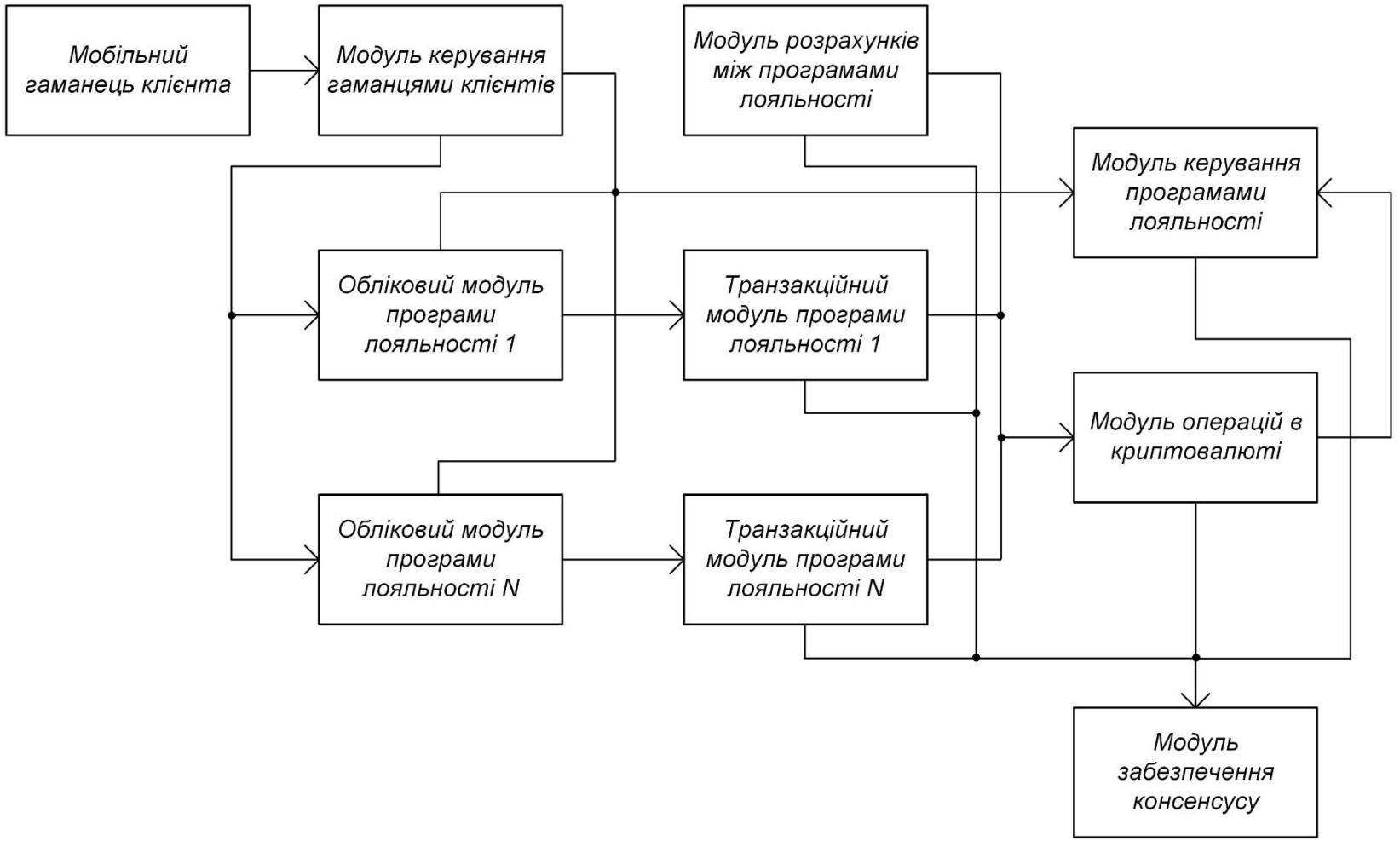
```
        lse.setState(LoyaltySystemExtrn.STATE_OK);
        ctx.getStub().putStringState(lsExtrn, lse.toJSONString());
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnBuy buyExtrnExec(Context ctx, String extrnId, String bonusAmount, String clientCard) {
        LoyaltySystemExtrn lse = loLsExtrnCheck(ctx);
        LoyaltySystem ls = lsGet(ctx, lse.getId());
        if (ls.getRate() == null || ls.getRate().getBuy() == null) throw new RuntimeException("Rate not defined.");
        long ccAmount = new BigDecimal(bonusAmount).setScale(ls.getRate().getScale(), RoundingMode.HALF_UP)
                .multiply(new BigDecimal(ls.getRate().getBuy()).setScale(ls.getRate().getScale(),
RoundingMode.HALF_UP))
                .setScale(0, RoundingMode.HALF_UP).longValue();
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract", "buyExec",
ls.getId(), "" + ccAmount, extrnId, bonusAmount);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        TransBuy trans = TransBuy.fromJSONString(rsp.getStringPayload());
        TransExtrnBuy transExtrn = new TransExtrnBuy();
        transExtrn.setId(ctx.getStub().getTxId());
        transExtrn.setTransCCId(trans.getId());
        transExtrn.setBonusAmount(bonusAmount);
        transExtrn.setBonusScale(ls.getRate().getScale());
        transExtrn.setCcAmount(trans.getCcAmount());
        transExtrn.setExtrnId(extrnId);
        transExtrn.setClientCard(clientCard);
        transExtrn.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setState(TransExtrnBuy.STATE_OK);
        ctx.getStub().putPrivateData(transExtrnBuyList, transExtrn.getId(), transExtrn.toJSONString());
        return transExtrn;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnBuy buyExtrnRefund(Context ctx, String transId, String stateMsg) {
        wmsCheck(ctx);
        TransExtrnBuy transExtrn = loBuyExtrnGet(ctx, transId);
        if (transExtrn.getState() != TransExtrnBuy.STATE_OK) throw new RuntimeException("Not refund.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract", "buyRefund",
transExtrn.getTransCCId(), stateMsg);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        transExtrn.setDtRefund(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setState(TransExtrnBuy.STATE_REFUND);
        transExtrn.setStateMsg(stateMsg);
        ctx.getStub().putPrivateData(transExtrnBuyList, transExtrn.getId(), transExtrn.toJSONString());
        return transExtrn;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnBuy buyExtrnGetOne(Context ctx, String transId) {
        TransExtrnBuy transExtrn = loBuyExtrnGet(ctx, transId);
        return transExtrn;
    }

    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnSell sellExtrnExec(Context ctx, String extrnId, String bonusAmount, String clientCard, String
transBuyId) {
        LoyaltySystemExtrn lse = loLsExtrnCheck(ctx);
        LoyaltySystem ls = lsGet(ctx, lse.getId());
        if (ls.getRate() == null || ls.getRate().getSell() == null) throw new RuntimeException("Rate not defined.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract", "sellExec",
ls.getId(), transBuyId, extrnId, bonusAmount);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        TransSell trans = TransSell.fromJSONString(rsp.getStringPayload());
        TransExtrnSell transExtrn = new TransExtrnSell();
        transExtrn.setId(ctx.getStub().getTxId());
        transExtrn.setTransCCId(trans.getId());
        transExtrn.setBonusAmount(bonusAmount);
        transExtrn.setBonusScale(ls.getRate().getScale());
        transExtrn.setCcAmount(trans.getCcAmount());
        transExtrn.setExtrnId(extrnId);
        transExtrn.setClientCard(clientCard);
        transExtrn.setDtCreate(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setDtExec(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setState(TransExtrnBuy.STATE_OK);
        transExtrn.setTransBuyId(transBuyId);
        ctx.getStub().putPrivateData(transExtrnSellList, transExtrn.getId(), transExtrn.toJSONString());
        return transExtrn;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnSell sellExtrnRefund(Context ctx, String transId, String stateMsg) {
        wmsCheck(ctx);
        TransExtrnSell transExtrn = loSellExtrnGet(ctx, transId);
```

```
        if (transExtrn.getState() != TransExtrnBuy.STATE_OK) throw new RuntimeException("Not refund.");
        Chaincode.Response rsp = ctx.getStub().invokeChaincodeWithStringArgs("CryptoCurrencyContract", "sellRefund",
transExtrn.getTransCCId(), stateMsg);
        if (rsp.getStatus() != Chaincode.Response.Status.SUCCESS) throw new RuntimeException(rsp.getMessage());
        transExtrn.setDtRefund(ctx.getStub().getTxTimestamp().atZone(ZoneId.of("UTC")));
        transExtrn.setState(TransExtrnBuy.STATE_REFUND);
        transExtrn.setStateMsg(stateMsg);
        ctx.getStub().putPrivateData(transExtrnSellList, transExtrn.getId(), transExtrn.toJSONString());
        return transExtrn;
    }
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public TransExtrnSell sellExtrnGetOne(Context ctx, String transId) {
        TransExtrnSell transExtrn = loSellExtrnGet(ctx, transId);
        return transExtrn;
    }
}

@DataType()
@Getter
@Setter
public class TransExtrnBuy {
    public static final int STATE_OK = 2;
    public static final int STATE_REFUND = 3;

    @Property()
    private String id;
    @Property()
    private String transCCId;
    @Property()
    private String bonusAmount;
    @Property()
    private int bonusScale;
    @Property()
    private long ccAmount;
    @Property()
    private String extrnId;
    @Property()
    private String clientCard;
    @Property()
    private ZonedDateTime dtCreate;
    @Property()
    private ZonedDateTime dtExec;
    @Property()
    private ZonedDateTime dtRefund;
    @Property()
    private int state;
    @Property()
    private String stateMsg;

    public String toJSONString() {
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.writeValueAsString(this);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }

    public static TransExtrnBuy fromJSONString(String json) {
        if(json==null) return null;
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.readValue(json.getBytes(StandardCharsets.UTF_8), TransExtrnBuy.class);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }
}

TransExtrnSell.java
package edu.doggy228.antoxasmart;

import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.Getter;
import lombok.Setter;
import org.hyperledger.fabric.contract.annotation.DataType;
import org.hyperledger.fabric.contract.annotation.Property;

import java.nio.charset.StandardCharsets;
```

```
import java.time.ZonedDateTime;

@DataType()
@Getter
@Setter
public class TransExtrnSell {
    public static final int STATE_OK = 2;
    public static final int STATE_REFUND = 3;

    @Property()
    private String id;
    @Property()
    private String transCCId;
    @Property()
    private String bonusAmount;
    @Property()
    private int bonusScale;
    @Property()
    private long ccAmount;
    @Property()
    private String extrnId;
    @Property()
    private String clientCard;
    @Property()
    private ZonedDateTime dtCreate;
    @Property()
    private ZonedDateTime dtExec;
    @Property()
    private ZonedDateTime dtRefund;
    @Property()
    private int state;
    @Property()
    private String stateMsg;
    @Property()
    private String transBuyId;

    public String toJSONString() {
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.writeValueAsString(this);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }

    public static TransExtrnSell fromJSONString(String json) {
        if(json==null) return null;
        try {
            ObjectMapper mapper = new ObjectMapper();
            return mapper.readValue(json.getBytes(StandardCharsets.UTF_8), TransExtrnSell.class);
        } catch (Throwable e) {
            throw new RuntimeException(e);
        }
    }
}
```

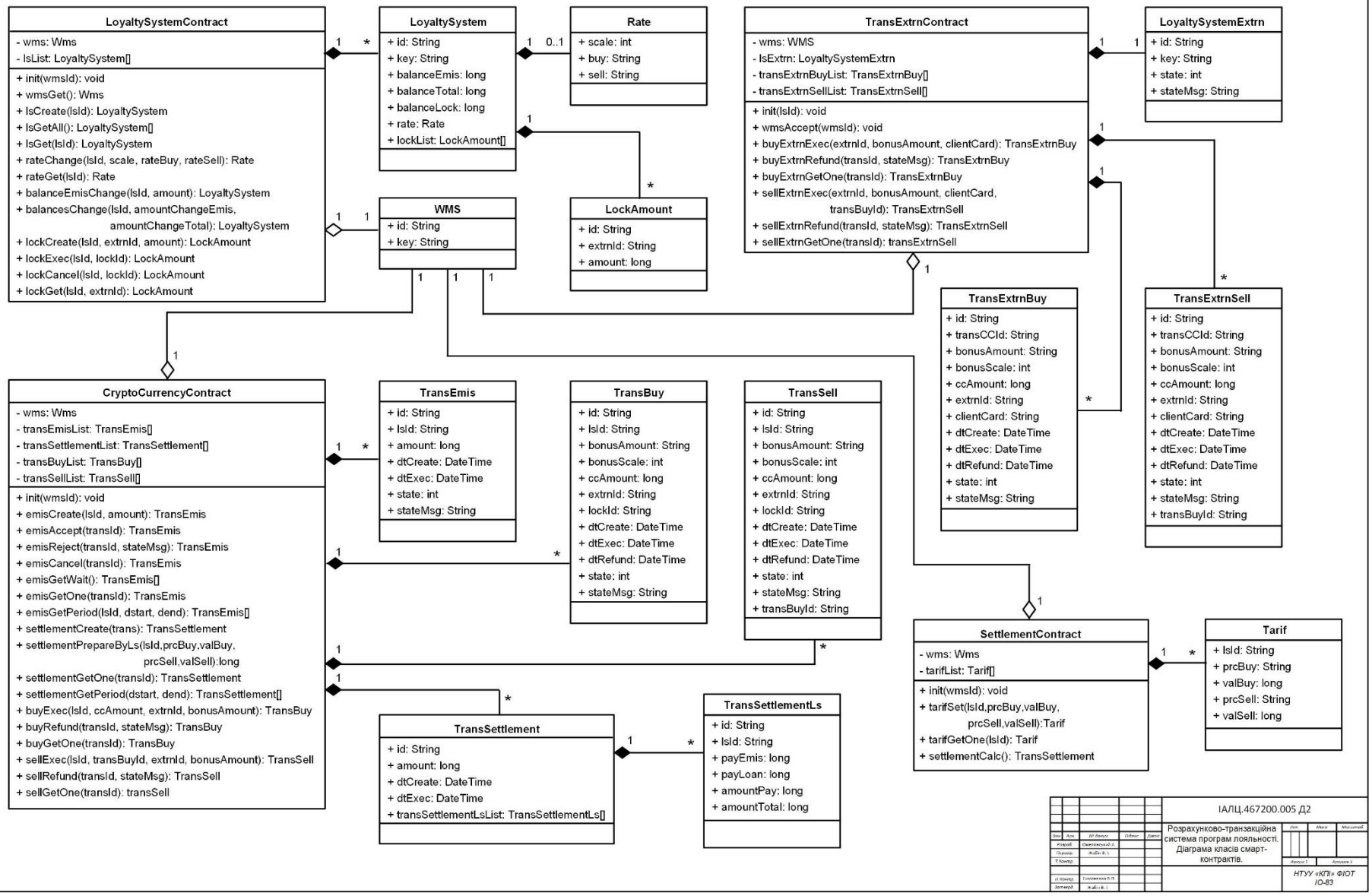
КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

Структурна схема системи



								ІАЛЦ.467200.004 Д1	
Змін.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Ємельянов Д.			Розрахунково-транзакційна система програм лояльності. Структурна схема системи.		Літ.	Маса	Масштаб
Перевір.		Жабін В. І.							
Т.контр.							Аркуш 1	Аркушів 1	
Н.контр.		Сологаєнко О.П.					НТУУ «КПІ» ФІОТ ІО-83		
Затверд.		Жабін В. І.							

Діаграма класів смарт-контрактів системи

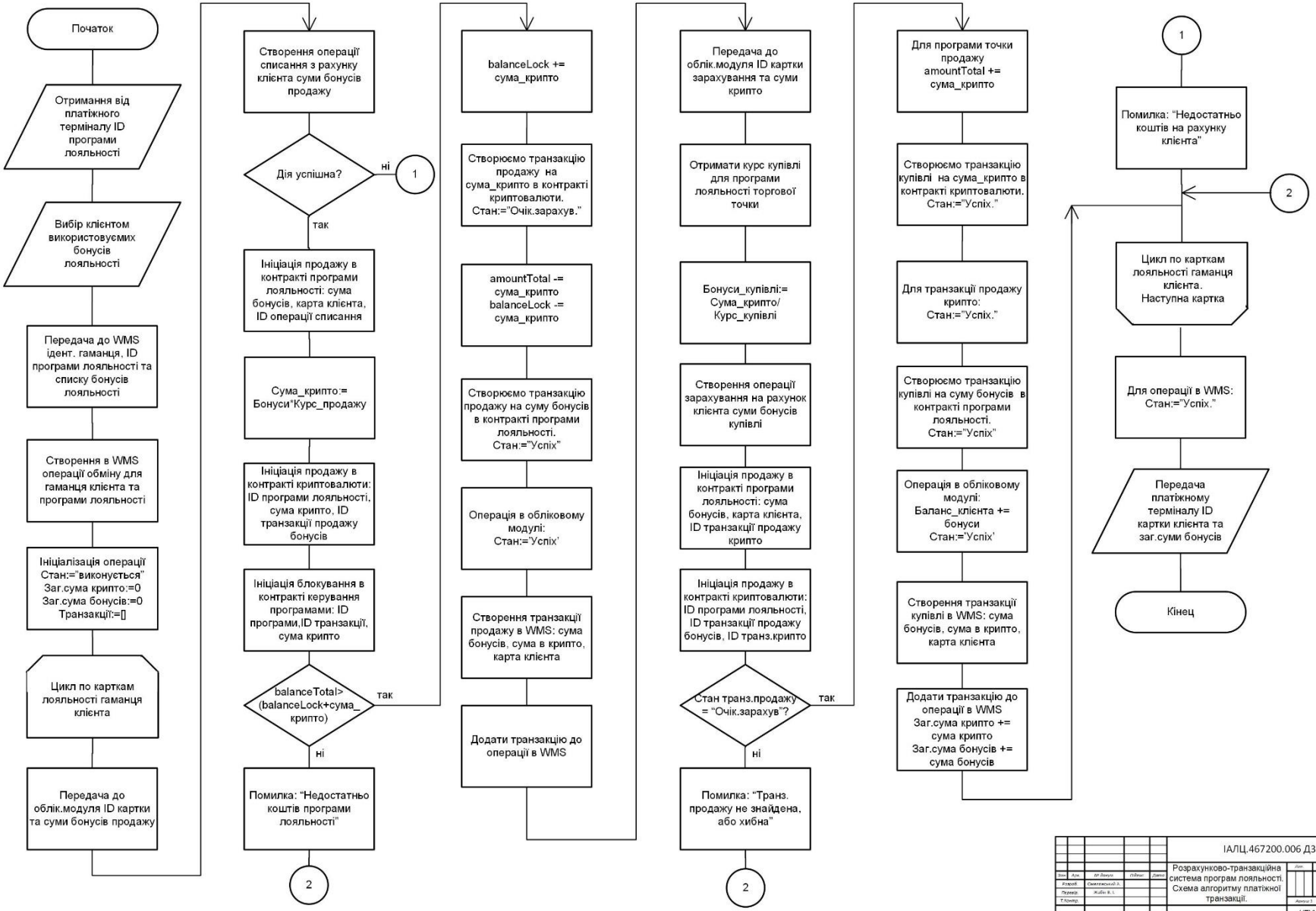


ІАЛЦ.467200.005 Д2

Розрахунково-транзакційна система програм лояльності. Діаграма класів смарт-контрактів.

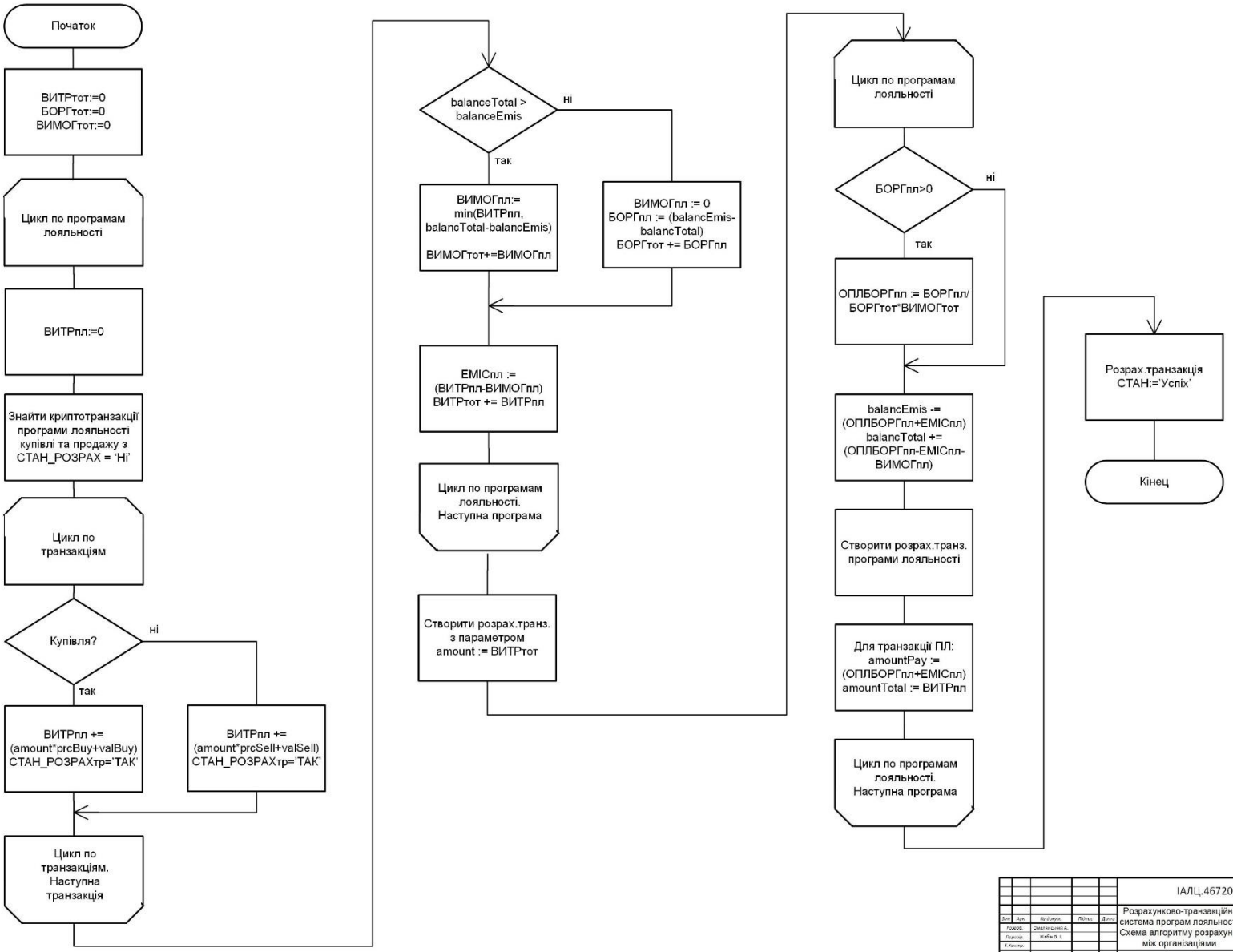
НТУУ «КПІ» ФІОТ ІО-83

Схема алгоритму платіжної транзакції



							ІАЛЦ.467200.006 ДЗ	
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.	Самолінський Л.						Розрахунково-транзакційна система програм лояльності. Схема алгоритму платіжної транзакції.	Літ. / Маса / Масштаб
Перевір.	Жабін В. І.							
Т.контр.								Аркуш 1 / Аркушів 1
Н.контр.	Сологенкіна Л.П.							НТУУ «КПІ» ФІОТ ІО-83
Затверд.	Жабін В. І.							

Схема алгоритму розрахунку між організаціями



						ІАЛЦ.467200.007 Д4	
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Омелянський А.			Розрахунково-транзакційна система програм лояльності. Схема алгоритму розрахунку між організаціями.	Літ.	Маса	Масштаб
Перевір.		Жабін В. І.						
Н.контр.						Аркуш 2	Аркушів 3	
Н.контр.		Сологаєнко О.П.				НТУУ «КПІ» ФІОТ ІО-83		
Затверд.		Жабін В. І.						