Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДППЗ.170111.01.11.ПЗ

Виконав студент IV курсу група ІПЗ-17-1 _____ А. В. Мельник
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Н. І. Праворська
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Г. І. Радельчук
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

7 червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Мельнику Андрію Васильовичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота
Керівник проєкту (роботи) Праворська Наталія Іванівна
кандидат педагогічних наук, доцент
Затверджена наказом ректора університету від 05.02.2021 р. № 11
2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2021 р.
3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмної системи
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 10 шт.)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г.І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І.В., доцент кафедри ІПЗ		

7. Дата видачі завдання «_05_» _лютого_ 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних тем ДП	01.12– 30.12.2020	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2021	
3 Проектування програмного забезпечення	01.02 – 28.02 2021	
4 Програмна реалізація	01.03 – 10.04.2021	
5 Тестування програмного забезпечення	11.04 – 30.04.2021	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2021	
7 Попередній захист ДП	травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2021	
9 Підготовка до захисту та захист ДП	з 01.06.2021	

Студент _____ А. В. Мельник
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Н. І. Праворська
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: «Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота».

Автор проекту: Мельник Андрій Васильович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 147 с., 42 рис., 7 табл., 4 дод., 16 джерел.

Графічна частина: 10 презентаційних слайдів.

КРИПТОВАЛЮТА, TRONIX, TRON, БЛОКЧЕЙН, ЕЛЕКТРОННИЙ ГАМАНЕЦЬ, REST, МІКРОСЕРВІСИ, SPRING, MONGODB, TELEGRAM, БОТ.

Метою проекту є розробка програмного забезпечення, яке дозволяє користувачу створювати гаманець криптовалюти Tronix, а також використовувати його за допомогою інтерфейсу у вигляді Telegram бота.

У дипломному проекті визначено особливості та відмінності криптовалют, переваги Tronix над іншими криптовалютами; проведений аналіз та недоліки існуючих рішень на ринку, аналіз Telegram як платформи для розробки; розроблена архітектура додатку на основі мікросервісів; проведений аналіз типів баз даних, розроблена структура даних.

Для реалізації програмного забезпечення використано мову програмування Java та фреймворк Spring, базу даних MongoDB.

В результаті проектування здійснена програмна реалізація електронного гаманця для криптовалюти Tronix з інтерфейсом у вигляді Telegram бота.

1.06.2021 р.
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДППЗ.170111.01.11.ПЗ	Пояснювальна записка	147		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні	10		
			слайди			

Змн.	Арк.	№ докум.	Підпис	Дата	ДППЗ.170111.01.11.ВД			
Виконав		Мельник А.В.		1.06	Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота	Літ.	Арк.	Аркушів
Керівник		Праворська Н.І.		2.06			1	1
Н. контр.		Радельчук Г. І.		3.06		ХНУ, ІПЗ-17-1		
Зав. каф.		Бедратюк Л.П.		4.06	Відомість документів			

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170111.01.11.ПЗ		
Виконав		Мельник А.В.		1.02	Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота	Літ.	Арк. 4 / Аркушів 147
Керівник		Праворська Н.І.		2.06			
Н. контр.		Радельчук Г. І.		3.06	Пояснювальна записка	ХНУ, ІПЗ-17-1	
Зав. каф.		Бедратюк Л.П.		4.06			

ВСТУП

У 2009 була створена перша криптовалюта у світі – Bitcoin [1], яка постала конкурентом банківській системі, використавши за основну технологію блокчейн. Даний вид цифрових валют побудований більш прозорим, надійним та безпечним шляхом в порівнянні з традиційними засобами, завдяки чому все більше людей починають цікавитись даним видом валюти та довіряти їй. Доказом цього є те, що станом на травень 2021 року, загальна капіталізація всіх криптовалют становить більше 2.3 трильйонів доларів США [2], додавши у вартості 300 мільярдів всього лише за 7 днів [3].

У 2017 році був створений TRON блокчейн з власною криптовалютою Tronix [4]. Даний проект зміг покращити ряд недоліків попередніх валют, надавши можливості розробки децентралізованих додатків на своїй платформі, швидший платіжний засіб в порівнянні з Bitcoin тощо. Завдяки своїм перевагам, дана електронна валюта є однією з найбільш використовуваних криптовалют.

Для використання криптовалют, були створені спеціальні додатки-гаманці, які дозволяють створювати нові акаунти в системі, виконувати перекази коштів, переглядати історію транзакцій у зручному вигляді тощо. Більшість гаманців для Tronix реалізовані у вигляді додатків для смартфонів, веб-додатків або плагінів для браузерів. Лише декілька варіантів такого ПЗ підтримують можливість використання різних версій у різних середовищах, проте жоден проект не надає можливості використовувати гаманець на всіх платформах.

Актуальність теми полягає в тому, що на сьогодні відсутня реалізація електронного гаманця для Tronix, який був би доступний одразу на всіх основних комп’ютерних платформах у різних виглядах з єдиним інтерфейсом, таких як: додаток для комп’ютера чи смартфону під різними операційними системами, веб-додаток з доступом через браузер тощо. З цієї причини, користування гаманцями з різних девайсів не є зручним для користувача, оскільки йому необхідно додавати свій гаманець на різні платформи та вивчати інтерфейси та функціонал

різних додатків. Дану проблему можна вирішити використавши платформу, яка надає можливість реалізації додатку для всіх платформ в єдиному вигляді – наприклад, Telegram надає можливість створити ботів з широким функціоналом.

Метою проекту є створення електронного гаманця для криптовалюти Tronix, який буде доступним на всіх основних комп'ютерних платформах за допомогою Telegram бота.

Завданнями роботи є:

– дослідити поняття блокчейну, його функціонування та особливості реалізації, особливості блокчейну TRON;

– дослідити платформу Telegram як базу для створення додатку;

– проаналізувати існуючі рішення на ринку, визначити їх недоліки та необхідний функціонал для даного типу ПЗ;

– побудувати архітектуру розроблюваної системи та обрати для розробки правильні інструменти та технології;

– реалізувати систему, яка задовольняє потрібний функціонал;

– провести повне тестування системи на наявність помилок та недоліків, відповідність вимогам.

Результатом дипломного проекту є функціонуюча система, яка надає користувачу можливість користуватись гаманцем валюти Tronix з будь-якого девайсу за допомогою Telegram бота.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Криптовалюта – це цифрова валюта, метою якої є створення альтернативи звичайним грошам. Інформація про належність монет (одиниця валюти) кожному власнику та їх переказ іншому користувачу зберігаються в обліковій книзі, яка існує у вигляді бази даних [5]. Створення нових монет (емісія), захист даних і функціонування системи досягаються за допомогою стійких криптографічних алгоритмів шифрування і цифрових підписів, та виконуються повністю автоматично, без адміністрування третіми особами.

В криптовалютах дані зберігаються децентралізовано, тобто кожен учасник системи може зберігати в себе репліку облікової книги зі всіма операціями в системі. Децентралізація стала основною особливістю цього виду цифрових валют, яка надала їм широкого розголосу та використання. Якщо ми говоримо про банківські системи – банк є центральним органом який відповідає за функціонування системи, її регулювання та збереження даних. Тобто, якщо в зловмисника вийшло підмінити дані в банку (наприклад, збільшити собі суму коштів на рахунку) то вся система буде працювати з вказаними змінами та вважати їх вірними. У випадку децентралізованої системи дії злочинця не зможуть нашкодити системі, оскільки дані зберігаються не в одному, а у багатьох місцях одночасно, і якщо зловмисник спробує використати підроблені дані – система відхилить операцію, оскільки в інших базах даних відсутній запис про наявність коштів у зловмисника.

Блокчейн (англ. Block chain, block – блок, chain – ланцюг) – реалізація децентралізованої бази даних, яка використовується криптовалютами [6]. Блокчейн зберігає дані у вигляді блоків, вибудуваних у певному порядку один за одним, утворюючи ланцюжок. При додаванні нових даних у блокчейн (таких як

транзакція-переказ монет, емісія тощо) інформація записується в базу даних у вигляді нового блоку, збільшуючи таким чином загальну довжину ланцюжка. Перед тим як додати блок у ланцюг, всі його транзакції перевіряються системою. Кожен блок окрім транзакцій містить в собі свій хеш, вхідними даними якого є дані для запису в цьому блоці, а також хеш попереднього блоку – таким чином підтверджуючи його та всі попередні блоки ланцюга. Виключенням з цього правила є перший блок в БД (genesis block), який не містить посилання на інший блок. Завдяки такій структурі даних, записану в блокчейні інформацію неможливо змінити (така можливість вкрай малоймовірна), оскільки зміна інформації в блоці також в результаті змінить хеш блоку, а в цьому випадку потрібно буде замінити і всі наступні блоки в системі, що вимагатиме великої кількості ресурсів.

Користувачів блокчейну можна поділити за наступними ролями: користувачі хто виконує перекази у системі, та ті хто ці перекази перевіряє на можливість виконання та достовірність, формує нові блоки даних з цих переказів та надсилає їх всім у системі (broadcasting). За виконану роботу учасники які перевіряють транзакції та створюють блоки отримують винагороду у вигляді монет та комісій за транзакції. Цих учасників у різних системах називають майнерами (miner), свідками (witness) тощо.

Виникає питання кому та за яких умов варто дозволити створювати нові блоки та, відповідно, отримувати за це винагороду. Проблема встановлення загальних правил задля збереження системи у робочому стані вирішується учасниками блокчейну прийняттям певного протоколу, тобто досягнення консенсусу. Ці умови, коли йде мова про розподілені системи такі як блокчейн, називають консенсус-протоколами. Метою такого протоколу є забезпечення погодженого стану даних після додавання нового блоку у ланцюг всіма користувачами, а також гарантувати що даний ланцюжок блоків є вірним.

Теперішніми найбільш популярними та розвинутими протоколами для використання криптовалютами є:

– доказ виконаної роботи – PoW (Proof-of-Work);

– підтвердження частки – PoS (Proof-of-Stake);

– делеговане підтвердження частки (Delegated Proof-of-Stake).

Доказ виконаної роботи – протокол, застосований у самих популярних криптовалютах – Bitcoin та Ethereum. Щоб додати новий блок до ланцюжка, майнер повинен довести що він виконав певний об'єм роботи [7]. Він вирішує важку задачу знаходження хешу, який відповідає певній умові. Перший, хто знайшов такий хеш, додає блок до ланцюга. Враховуючи обчислення складних математичних задач, даний протокол є витратним в плані обчислювальних ресурсів та електроенергії. Спеціалісти намагались створити алгоритм, при якому дані обчислення вирішували б певні математичні проблеми з реального світу, але повноцінно цього реалізувати так і не вдалось. Ще одним недоліком PoW рахується те, що математичні задачі для додавання блоку займають багато часу, тому пропускна здатність блокчейну з таким протоколом є низькою. Наприклад, Bitcoin генерує один блок в 10 хвилин. Це приводить до перенавантаження системи транзакціями, які треба виконати. Задля того щоб транзакція швидше потрапила до блоку, користувачі пропонують за них більші комісії майнерам, в результаті чого загальна ціна транзакцій зростає. 22 лютого 2021 року комісії в блокчейні Ethereum зросли на стільки, що Binance, одна з найбільших криптовалютних бірж, призупинила весь потік криптовалюти Ethereum з біржі [8]. Також, з таким підходом до розподілу нагород заробляти монети можуть й ті, кого цікавить лише власна вигода, а не цілісність системи. Перевага цього протоколу – кожен може приєднатись до пошуку рішення задач задля отримання винагороди, тобто цей протокол може працювати за умов, коли ніхто з користувачів нікому не довіряє.

Підтвердження частки – протокол, особливістю якого є відсутність непотрібних та важких обчислень. Щоб отримати можливість створити новий блок, користувачі повинні внести заставу у вигляді своїх токенів [1]. Алгоритм протоколу обирає одного з учасників, які внесли заставу щоб згенерувати

наступний блок, і чим більшу заставу зробив користувач, тим більша ймовірність, що він буде обраним. Якщо користувач не створив блок, його створить інший користувач. Особливість цього протоколу полягає в тому, що користувачі, які хочуть отримати нагороду, зацікавлені в безпеці системи, оскільки вони зобов'язані мати у власності монети. Якщо вони заподіють шкоду блокчейну – вони цим самим нанесуть шкоду й самим собі. Перевагою цього протоколу вважається більша швидкість в порівнянні з PoW. Недоліком системи виступає стимуляція учасників до накопичення коштів в одних руках, що може негативно позначитися на децентралізації мережі – багатші учасники будуть швидше ставати багатими. Таким самим чином, якщо група осіб буде мати в себе значну суму коштів – вони можуть нав'язувати свої умови функціонування та шляхи розвитку системи.

Делеговане підтвердження частки – протокол, при якому учасники блокчейну делегують створення нових блоків певному фіксованому числу користувачів (свідків) [9]. Користувачі голосують за кандидатів на цю позицію, вносячи заставу у вигляді монет. Чим більше монет було внесено під заставу, тим більше голосів буде начислено кандидату. Користувач може проголосувати в будь-який час. Користувачі отримують токени від того свідка, за якого вони проголосували. За рахунок меншої кількості осіб які створюють блоки, блоки створюються швидше ніж у попередніх протоколів, за рахунок чого збільшується пропускна здатність блокчейну. Даний протокол є найбільш справедливим по відношенню до всіх користувачів блокчейну, оскільки кожен користувач може впливати на вибір свідків для створення блоків. Мінусом протоколу є можливість централізації завдяки обмеженій кількості свідків, але якщо свідок погано справляється зі своєю роботою, користувачі замінять його іншим свідком.

Основною одиницею в блокчейні, яка забезпечує цілісність та функціонування системи є вузол (Node) – спеціальне програмне забезпечення, яке відповідає за збереження даних блокчейну, їх перевірку, та передачу даних іншим вузлам. Коли користувач хоче створити транзакцію, дізнатись баланс

певної адреси чи інші параметри блокчейну – він повинен надіслати вузлу запит на виконання тієї чи іншої операції. Щоб вузол функціонував без помилок та з актуальними даними, він повинен бути постійно під'єднаним до мережі, щоб мати можливість записувати нові блоки в свою базу даних.

Іноді в блокчейні виникає ситуація коли декілька, щойно створених в різних частинах мережі, блоків посилаються на один і той самий блок, створюючи таким чином різні гілки ланцюжка блоків. Ці гілки можуть містити різні операції з одними й тими самими акаунтами та коштами користувачів, а тому така ситуація є небезпечною для системи. В такому разі, кожен учасник блокчейну зберігає та записує кожну з гілок, допоки не буде відновлений зв'язок різних частин мережі та не відбудеться обмін гілками всіма учасниками. Коли такий зв'язок відновлено, з поміж наявних гілок обирається головна. Зазвичай, головною гілкою вважається гілка з найдовшим ланцюжком блоків, або гілка для якою було затрачено найбільше обчислень.

Tronix, також відома як TRX – криптовалюта, побудована в екосистемі TRON – децентралізованій операційній системі, заснованій на блокчейні у 2017 році. Один TRX можна розділити до 5 знаків після коми. 0.000001 частинка TRX називається SUN (в честь засновника). Тобто, 1 TRX = 1000000 SUN. На сьогоднішній день, криптовалюта входить до топ-100 криптовалют з найбільшою ринковою капіталізацією. Tronix працює на протоколі делегованого підтвердження частки, обираючи 27 свідків (яких в системі називають Super Representatives, SR, тобто супер представники) кожних 6 годин. Супер представники відповідають не лише за створення блоків та безпеку блокчейну, а й за визначення подальшого шляху розвитку криптовалюти. Якщо в TRON блокчейні з'явились декілька гілок ланцюжка блоків, правильною вважається найдовша гілка. Кожен доброчесний SR намагається додати блок до найдовшої гілки. Враховуючи виникнення такого гілкування, кожен новий блок в ланцюжку спочатку вважається непідтвердженим. Статус затвердженого він набуває лише тоді, коли більш ніж 70% SR затвердили його, тобто продовжили ланцюжок на

основі вказаного блоку. Таким чином, блок стає затвердженим тоді, коли на його основі були побудовані більше 18 наступних блоків (27 SR * 70% = 18.9 блоків).

У 2020 році було проведене дослідження, яке показало що максимальна пропускна здатність блокчейну Tronix є близько 700 транзакцій в секунду [10], що в рази перевищує показники найвідоміших криптовалют Bitcoin та Ethereum (3 та 25 транзакцій в секунду відповідно). Завдяки використанню сучасного та швидкого консенсус-протоколу, Tronix є криптовалютою з великим потенціалом та кількістю побудованих систем навколо власного блокчейну.

Екосистема TRON використовує систему акаунтів, при якій для кожного користувача генерується акаунт (приватний ключ) – випадкове число, яке генерується за допомогою криптографічного алгоритму ECDSA з еліптичною кривою SECP256K1. Цим ключем користувач підписує свої транзакції та інші операції, чим підтверджує свою особу при оперуванні коштами акаунту. Щоб отримати кошти, користувач повинен надати свою адресу, яка в свою чергу є відформатованим публічним ключем приватного ключа (публічний ключ завжди можна отримати з приватного ключа). Тобто, користувачу необхідно володіти лише приватним ключем, який надає можливість керувати коштами акаунта. Окрім власного балансу, кожен акаунт має наступні ресурси, які використовуються в системі: бали пропускної здатності (Bandwidth points) та енергія (Energy) [11]. Енергія використовується для виконання різноманітних операцій в екосистемі TRON, у роботі з Tronix вона не приймає участі. Бали пропускної здатності використовуються для створення транзакцій та нових акаунтів у системі, тобто виступають в ролі комісії. За кожну транзакцію, користувач витрачає стільки балів пропускної здатності, скільки місця в блокчейні займає інформація про транзакцію (в байтах). Кожен акаунт має початкові 5000 балів пропускної здатності, які відновлюються щодня. Також, користувач може внести заставу в монетах Tronix (заморозити свої кошти), щоб отримати додаткові бали. Кількість отриманих балів дорівнює частці суми Tronix, яку користувач вирішив заморозити від загальної кількості заморожених коштів

в системі всіма користувачами помножених на сталий коефіцієнт (його значення та загальна сума застави всіх користувачів зберігаються в блокчейні). Внесені під заставу кошти можна буде розморозити лише після 3 днів з останньої застави. Якщо в користувача недостатньо балів, щоб виконати переказ коштів, у вигляді комісії виступає сама криптовалюта Tronix, яка списується з акаунта. Списана сума TRX дорівнює необхідній кількості балів пропускної здатності помножених на 140 SUN. Після генерації нового акаунта, його потрібно активувати, виконавши переказ на його адресу. До активації, гаманець вважається неактивним, та не буде доступний для перегляду у блокчейні.

Кожна транзакція містить наступну інформацію, яку необхідно знати користувачу: блок у якому знаходиться транзакція, хеш (ідентифікатор) транзакції, тип транзакції (переказ, застава тощо), адрес відправника та отримувача (у випадку переказу), сума TRX, дата створення транзакції, комісія (кількість затрачених балів пропускної здатності або списаних TRX), чи підтверджений блок з даною транзакцією, її статус (успішна, невдала, відхилена блокчейном тощо) та можлива текстова примітка (до 200 символів).

Важливим питанням при роботі з криптовалютою є спосіб зберігання та використання коштів. Задля автоматизації звернень до вузлів блокчейну з метою виконати певні операції, було створене спеціальне програмне забезпечення, яке спрощує цю взаємодію – електронний гаманець. Таке ПЗ полегшує роботу користувачу, оперуючи його приватним ключем та адресом гаманця при створенні вихідних транзакцій, зчитуванні вхідних транзакцій з блокчейну тощо. Всі електронні гаманці можна поділити на 2 категорії: холодні (cold wallet) та веб-гаманці (також називають гарячими) [12]. Електроні гаманці створюють акаунт у блокчейні для користувача самостійно. Певні гаманці також реалізують можливість імпортувати вже створений гаманець у вигляді приватного ключа.

Холодний гаманець – програма, яка встановлюється на комп’ютер або телефон користувача, та зберігає в собі всю необхідну інформацію про його акаунт. Іноді представлений у вигляді спеціалізованого програмно-апаратного

комплексу, який має переваги у вигляді додаткового шифрування даних (наприклад, Ledger nano X). Головна перевага такого типу гаманця – приватний ключ зберігається на вашому пристрої, і лише власник знає його значення. Його основний недолік – те, що взаємодія з ресурсами гаманця можлива лише при наявності доступу до пристрою. Відповідно, якщо втратити пристрій – дані та кошти гаманця можуть бути назавжди втрачені.

Більш поширеним варіантом електронного гаманця є веб-гаманець – гаманець, доступ до якого можна отримати онлайн, тобто в мережі. Зазвичай реалізовані у вигляді веб-сайтів чи розширень для браузера, такі гаманці є менш безпечними, оскільки доступ до них виконується за допомогою інтернету. Однак, з вказаного мінусу витікає його головна перевага – можливість використати свій гаманець онлайн з любого девайсу, який має доступ до мережі.

Основна проблема веб-гаманців – те, що розробка такого ПЗ зазвичай ведеться для певних платформ, наприклад, мобільних пристроїв під керуванням ОС Android або iOS. У випадку, якщо користувач захоче скористатись коштами свого гаманця з комп’ютера чи браузера, йому необхідно буде встановлювати електронний гаманець від іншого розробника, вчитись користуватись іншим інтерфейсом та створювати ще один окремий обліковий запис. Також, гаманці які написані під різні платформи можуть оновлюватись з різними часовими проміжками. Тобто, якщо в мобільній версії гаманець отримав новий функціонал, він може бути відсутнім протягом певного часу у веб-версії.

Задля вирішення цієї проблеми, необхідно використати платформу, яка існує у вигляді додатків для мобільних пристроїв, комп’ютерів та веб-додатку у браузері, з можливістю одночасного оновлення всіх реалізацій. Такі можливості надає месенджер Telegram, який станом на 2021 рік є найшвидшим засобом спілкування з власним протоколом захисту спілкування MTProto [13]. Даний протокол забезпечує захист повідомлень, дзвінків та відеозв’язку краще, ніж його прямі конкуренти, такі як WhatsApp, Facebook Messenger, Viber тощо. Telegram входить до 20 найбільших соціальних платформ за кількістю користувачів, та

налічує понад 500 мільйонів активних користувачів [14]. Telegram дозволяє розробникам ПЗ створювати ботів (спеціальні програми, які можуть виконувати певні команди користувачів), та інтегрувати їх в свій месенджер у вигляді звичайного чату, який ми бачимо при спілкуванні з іншою людиною. Особливість реалізації даних ботів у тому, що Telegram надає веб інтерфейс (веб API), завдяки якому розробник може отримувати нові повідомлення та команди від користувача, та надсилати повідомлення йому у відповідь. Тобто, реалізація обробки повідомлень та відповідей на ці повідомлення лежить на розробнику бота. Боти можуть відображати в переписці з користувачем різноманітні кнопки, що дозволяє використовувати їх, як звичайний інтерфейс додатку. Враховуючи, що боти працюють однаково на всіх платформах, достатньо створити лише одну реалізацію гаманця, яка буде працювати як з мобільними версіями месенджера, так і з іншими. Таким чином, оновлення гаманця будуть випускатись на всіх платформах одразу, що є вагомим аргументом при виборі платформи.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Розглянемо існуючі варіанти електронних гаманців для Tronix від провідних розробників, якими користуються найбільша кількість користувачів, щоб визначити основні переваги та виокремити недоліки, на які можна спиратись при розробці та встановленні функціональних вимог.

TokenPocket – електронний гаманець для криптовалют, який дозволяє створити акаунти для багатьох валют, таких як Bitcoin, Ethereum, Tronix тощо, та користуватись ними в одному додатку. Гаманець працює на платформах Android, iOS, MacOS, Windows. Процес реєстрації активного гаманця, коли користувач може імпортувати гаманець вже існуючий гаманець ввівши свій приватний ключ, або створити новий гаманець зображений на рисунку 1.1.

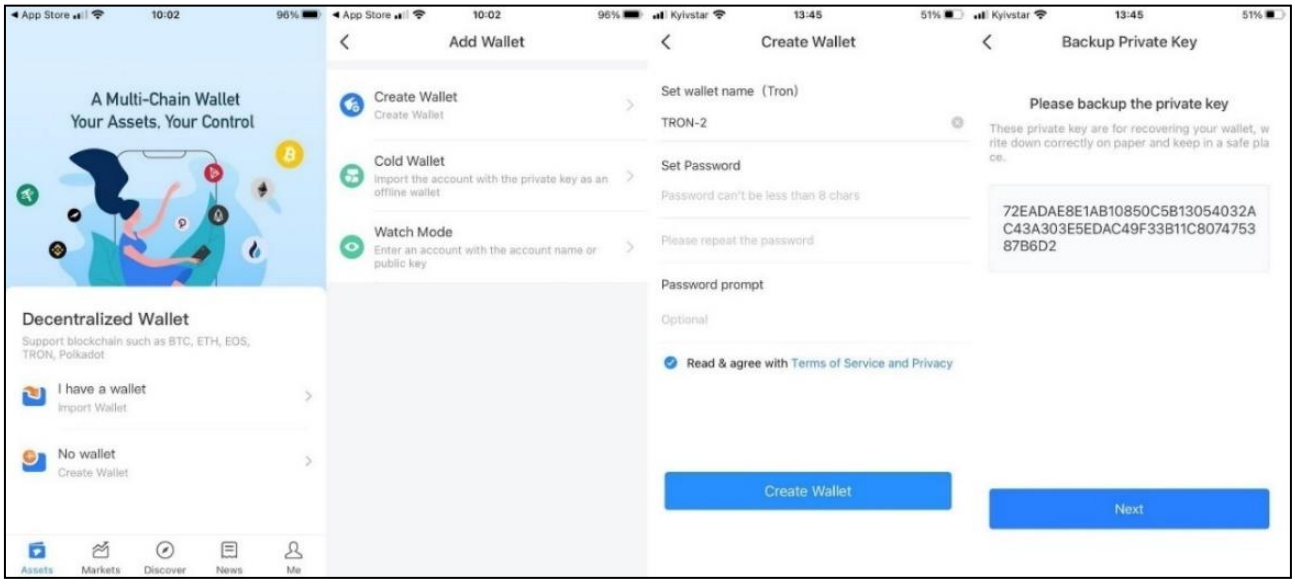


Рисунок 1.1 – Реєстрація гаманця у TokenPocket

Створення нового гаманця вимагає від користувача створити пароль для гаманця та його назву, після чого користувач має зберегти собі приватний ключ, та підтвердити його. Щоб експортувати приватний ключ гаманця (або просто його побачити), користувач повинен буде ввести пароль, створений раніше. Також, TokenPocket дозволяє подивитись баланси гаманців інших користувачів за їх адресами. У гаманці користувач має можливість зареєструвати декілька акаунтів в блокчейні. Користувач також може змінити пароль та додати відбиток пальця для автентифікації.

Вікна гаманця які показують баланс користувача, меню з можливістю експортувати приватний ключ та змінити пароль, вікно для контролю ресурсів з можливістю заморозити монети для збільшення балів пропускної здатності зображені на рисунку 1.2. Історія транзакцій гаманця, вікно створення нової транзакції та вікно з деталями транзакції зображені на рисунку 1.3. Перевагами гаманця є зручний інтерфейс, можливість встановити відбиток пальця замість пароля, створити гаманці для інших криптовалют. До недоліків можна віднести відсутність гаманця на платформах Linux та веб-додатку, а тому буде необхідність використовувати додаткові гаманці.

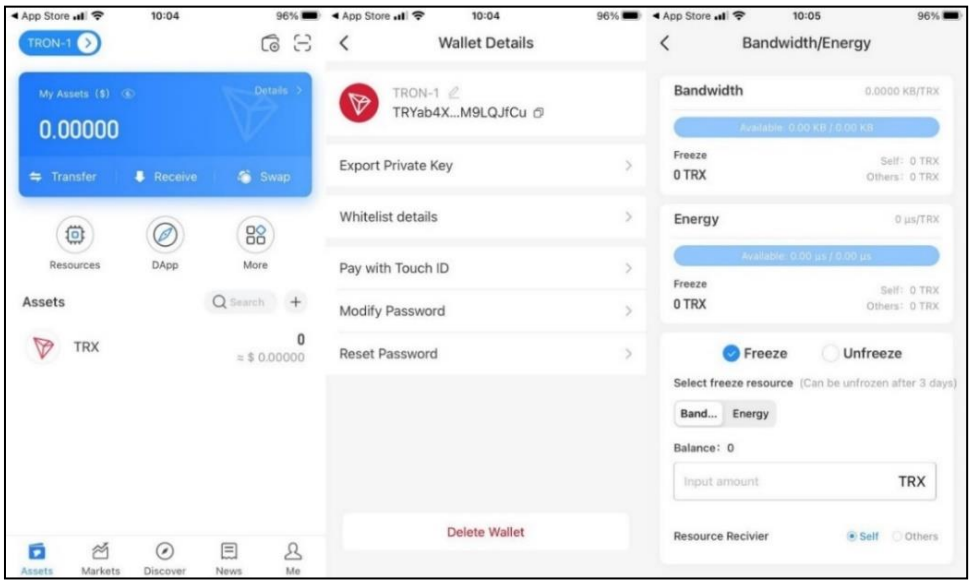


Рисунок 1.2 – Огляд гаманця, керування ресурсами гаманця у TokenPocket

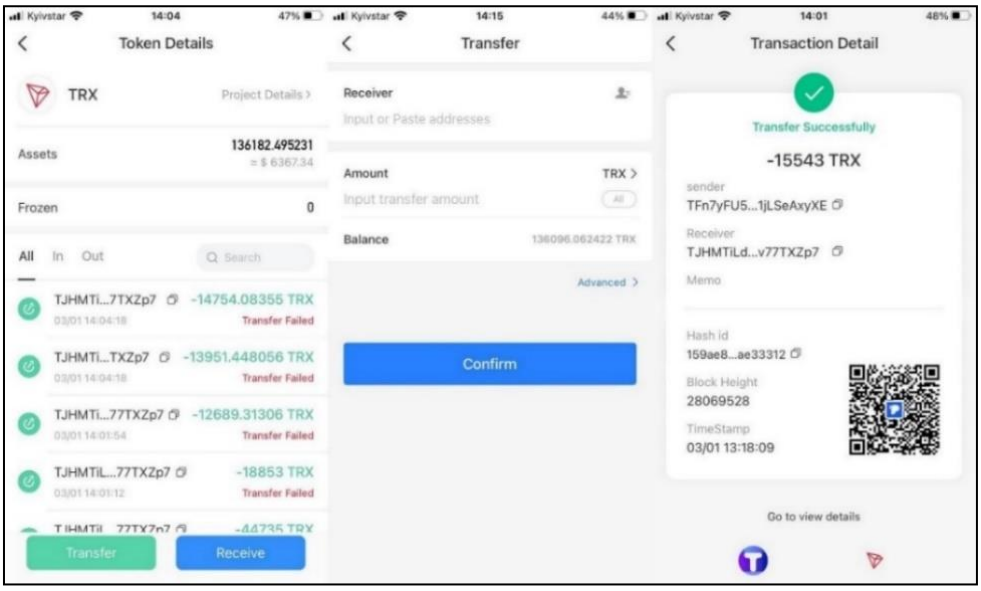


Рисунок 1.3 – Перегляд та створення транзакцій у TokenPocket

TronLink – електронний гаманець для Tronix, який розробляється розробниками TRON, завдяки чому він є найпопулярнішим гаманцем для обраної криптовалюти. Існує у вигляді додатків для Android, iOS, а також розширення до браузера Google Chrome. При створенні гаманця, як і в TokenPocket, також необхідно створити пароль та дати назву гаманцю. Головне вікно версії гаманця для браузера, де можна побачити баланс гаманця, ресурс енергії, балів пропускної здатності, інших валют платформи TRON зображене на рисунку 1.4.

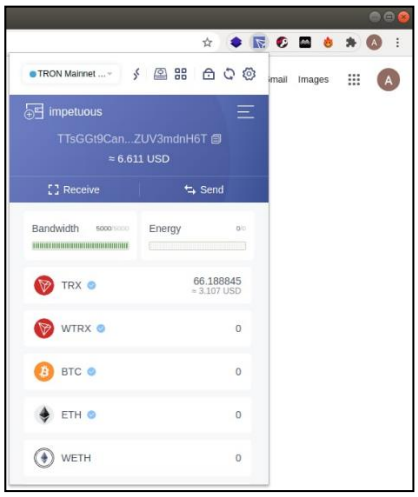


Рисунок 1.4 – Огляд гаманця TronLink

Інтерфейс для роботи з транзакціями, ідентичний інтерфейсу гаманця TokenPocket зображений на рисунку 1.5.

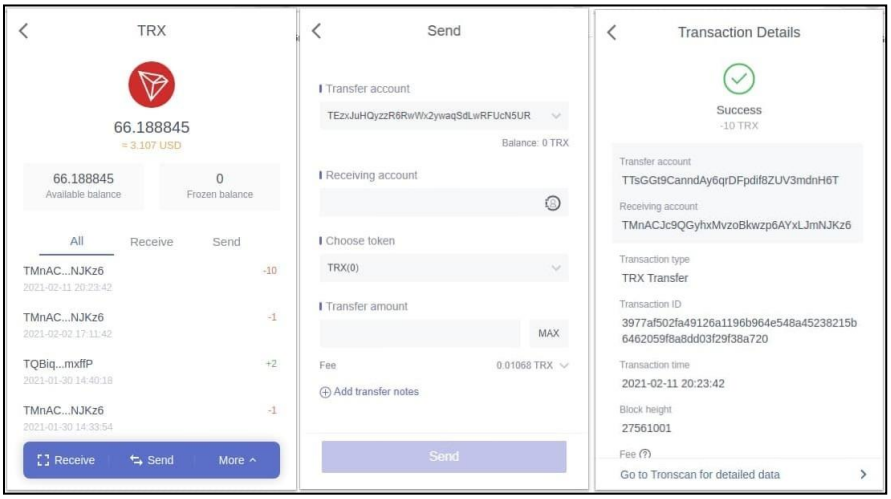


Рисунок 1.5 – Перегляд та створення транзакцій у TronLink

Головною перевагою цього гаманця є те, що його розвивають розробники платформи TRON, тобто він один з перших містить найактуальніші оновлення по взаємодії з блокчейном. Як і гаманець TokenPocket, гаманець показує наявний баланс користувача, конвертований у звичні валюти (в даному випадку, долар США) за актуальним курсом валют, що є також корисним для підрахунку балансу. Недоліком виступає те, що для застави коштів, розширення перенаправляє користувача на інший веб-додаток, що є не зручним та може

спантеличити користувача. Також, гаманець одразу відображає багато валют платформи TRON, навіть якщо вони не використовуються користувачем – це надлишкова інформація, яка більшості користувачів не потрібна. Гаманець не сповіщає власника, коли на баланс зачислені нові кошти. Провівши аналіз існуючих рішень, можна виокремити наступні функції, якими повинен володіти електронний гаманець для Tronix:

– можливість створити, імпортувати та експортувати гаманець;

– перегляд балансу як свого гаманця, так і гаманців інших учасників;

– пошук транзакції за її ідентифікатором;

– перегляд історії транзакцій власного гаманця;

– відображення балансу користувача в еквіваленті звичних валют, таких як гривня, долар США та відображення курсу TRX до валют;

– отримання сповіщень про поповнення балансу;

– можливість створення транзакцій (переказу коштів, застава коштів, відміна застави) з попереднім переглядом вартості переказу.

1.3 Аналіз вимог до програмного забезпечення та розробка технічного завдання

Проаналізувавши предметну область та доступного програмно-технічного забезпечення, необхідно визначити та описати вимоги до розроблюваного ПЗ. Для цього використаємо мову UML – мова візуального моделювання, яка розроблена для специфікації, візуалізації, проектування і документування компонентів програмного забезпечення. В рамках мови UML всі представлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, які отримали назву діаграм.

Діаграма варіантів використання – діаграма, на якій зображуються відношення між акторами та варіантами використання. Діаграма варіантів

використання є вихідним концептуальним представленням або концептуальною моделлю системи в процесі її проектування та розробки.

Складемо описи користувачів системи (акторів) та необхідного їм функціоналу та сервісів (варіанти використання). У таблиці 1.1 наведені актори розроблюваного програмного комплексу.

Таблиця 1.1 – Опис акторів розроблюваного ПЗ

Актор	Короткий опис
Користувач без активного гаманця	Може переглядати баланси та інші ресурси гаманців, шукати та переглядати транзакції в блокчейні, переглянути ціну TRX в гривні та доларі США. Може створити або імпортувати власний гаманець.
Користувач з активним гаманцем	Після створення або імпортування гаманця, користувач може виконувати всі дії по перегляду блокчейну, які міг виконувати до цього, а також створювати транзакції, залишати токени під заставу, виводити їх з під застави та експортувати свій гаманець.

В таблиці 1.2 наведені варіанти використання розроблюваного ПЗ.

Таблиця 1.2 – Опис варіантів використання розроблюваного ПЗ

Актор	Найменування ВВ	Опис ВВ
1	2	3
Користувач без активного гаманця	Створення нового гаманця	Користувач створює новий гаманець, встановлює пароль для його експорту.
	Імпорт гаманця	Користувач імпортує власний гаманець в систему за допомогою приватного ключа. Попередньо також необхідно встановити пароль для гаманця.

Кінець таблиці 1.2

1	2	3
Будь-який користувач	Перегляд балансу гаманців	Користувач може переглянути баланс гаманця, ввівши його адресу.
	Пошук транзакції	Користувач може знайти відомості про транзакцію, ввівши її ідентифікатор.
	Перегляд курсу Tronix до інших валют	Користувач може побачити актуальний курс TRX до інших валют.
Користувач з активним гаманцем	Застава TRX для ресурсів	Користувач вносить заставу у вигляді TRX, щоб отримати бали пропускної здатності.
	Створення транзакцій	Користувач переводить гроші на інший гаманець, вказавши адресу та суму переказу.
	Експорт гаманця	Користувач отримує приватний ключ гаманця, та стирає його в системі (опційно)
	Перегляд історії транзакцій	Користувач переглядає історію транзакцій по активному гаманцю
	Отримання сповіщень про поповнення балансу	Користувач отримує сповіщення про нові вхідні транзакції

Визначивши акторів системи та варіанти використання, побудуємо діаграму варіантів використання (рисунок 1.6).

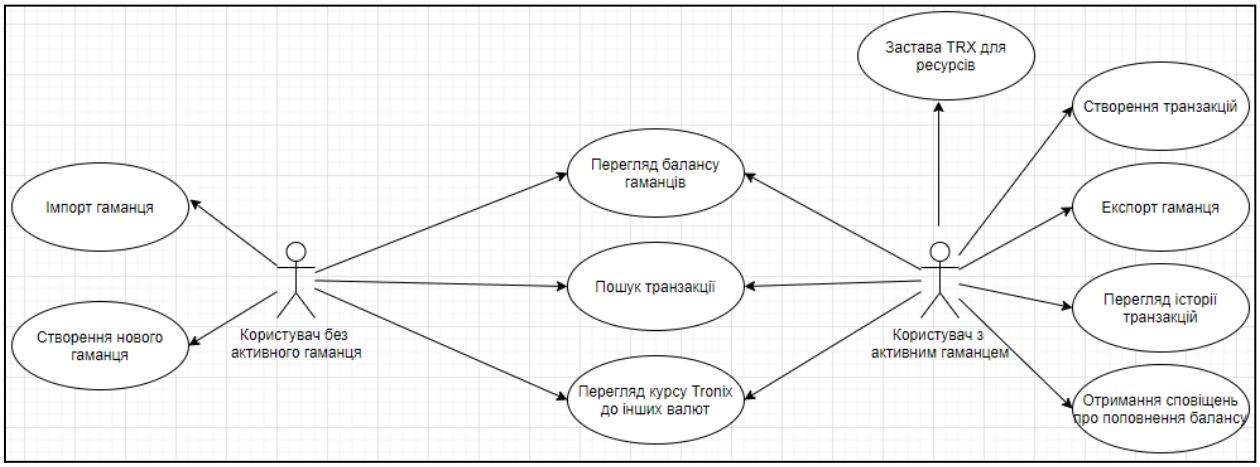


Рисунок 1.6 – Діаграма варіантів використання розроблюваного ПЗ

На основі проведеного аналізу вимог до ПЗ було розроблене технічне завдання, яке подано у додатку А.

У цьому розділі було проведено аналіз та огляд технології блокчейн, проаналізовано її основні реалізації та основні особливості. Було проведено аналіз криптовалюти Tronix, її специфіки та основні відмінності й переваги від інших криптовалют. Були розглянуті різні види електронних гаманців, проведений аналіз існуючих рішень та проблем, огляд можливостей Telegram як платформи для розробки ботів, в результаті чого були визначені функціональні вимоги до розроблюваного програмного забезпечення.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕНЯ

2.1 Аналіз та проектування архітектури системи

Згідно з поставленою задачею, інтерфейсом користувача є бот у Telegram месенджері. Як було вказано раніше, вся взаємодія системи з користувачем використовуючи бота буде відбуватись за допомогою веб API, використовуючи яке система отримує нові повідомлення та надсилає відповідь. Завдяки вказаній взаємодії за допомогою веб-мережі, розроблювана система є веб-додатком. Враховуючи те, що телеграм бот є інтерфейсом користувача, а розроблюване ПЗ повинне обробляти вхідні повідомлення до бота, така система є реалізацією клієнт-серверної архітектури.

Клієнт-серверна (дворівнева) архітектура – один з видів архітектури ПЗ, при якому на найвищому рівні виділяються два основні абстрактні елементи – клієнт та сервер. Клієнтом називають ПЗ, яке виконує запити до серверу, з метою отримати певні результати. Клієнтом можуть виступати додатки будь-якого вигляду: плагін для браузера, додаток на телефоні, чи фоновий процес запущений на персональному комп’ютері. Сервером виступає ПЗ, яке отримує та обробляє запити клієнтів повертаючи необхідний результат у відповідь. На сьогодні, є два основні види комунікації між клієнтом та сервером: комунікація між процесами, та комунікація за допомогою комп’ютерних мереж. Комунікація між процесами використовується в межах одного комп’ютера, що не підходить нам, тому що клієнтами є будь-який девайс з доступом до глобальної мережі інтернет. В випадку нашої системи, комунікація буде відбуватись за допомогою веб-мережі, а тому модель даної системи можна зобразити рисунком 2.1.

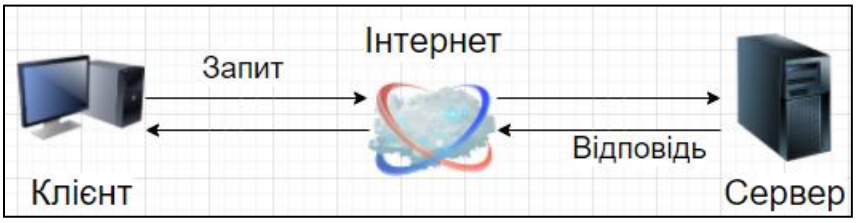


Рисунок 2.1 – Модель клієнт-сервер

Розглянемо та виберемо спосіб комунікації між клієнтом та сервером. Для комунікації використовуючи інтернет, використовують різноманітні протоколи передачі даних. Найбільш поширеними актуальними протоколами можна назвати WebSocket та HTTP.

WebSocket – протокол, призначений для комунікації між клієнтом та сервером за допомогою двостороннього каналу зв'язку. За рахунок постійно відкритого каналу між програмами, WebSocket забезпечує обмін інформацією між сервером та клієнтом у режимі реального часу. Даний протокол часто використовують при потребі швидкого оновлення інформації – соціальні мережі, валютні біржі, чати тощо.

HTTP – протокол, який застосовується для передачі даних у вигляді запит-відповідь, тобто для кожного запиту клієнт створює новий сеанс зв'язку з сервером. Варто відмітити, що протокол WebSocket для встановлення зв'язку з сервером також використовує HTTP. Початково, протокол використовувався для передачі веб-сторінок, але сьогодні його використовують для передачі як файлів, так і інформації в іншому вигляді. Даний протокол краще підходить для нашої системи, оскільки кожен запит користувача (переказ коштів, перегляд транзакцій) – це окрема короткотривала дія, в якій відсутня необхідність постійного зв'язку. Даний протокол буде використовуватись як для комунікації з блокчейном, так і з веб API Telegram.

Кожен HTTP запит містить в собі наступну інформацію: рядок запиту, поля-заголовки (headers). Також, якщо клієнт хоче передати дані серверу, такі як файл або персональні дані для реєстрації, запит містить тіло з вказаною інформацією. Поля-заголовки містять в собі різну додаткову інформацію та метадані для запиту, такі як мова користувача, у якому форматі надіслано тіло запиту, який формат очікується у відповідь тощо. Рядок запиту містить в собі уніфікований ідентифікатор ресурсу (URI), а також HTTP-метод, який вказує що саме необхідно зробити з даним ресурсом. Правильне використання цих методів дозволяє покращити розуміння системи, а також спрощує входження нових

розробників у процес розробки. Розглянемо основні HTTP-методи:

– GET – використовується для отримання ресурсу (наприклад, отримання історії транзакцій);

– POST – використовується для додавання нового ресурсу, такого як файл, публікація, новий переказ коштів тощо;

– PUT – використовується для оновлення ресурсу, наприклад мови сповіщень користувача;

– PATCH – використовується для часткового оновлення ресурсу, коли не потрібно змінювати весь ресурс;

– DELETE – використовується для видалення ресурсу (електронного гаманця, файлу тощо);

– OPTIONS – використовується для опису параметрів з'єднання з обраним ресурсом на сервері.

При розробці мережевого спілкування системи варто прийняти до уваги архітектурний підхід (стиль) взаємодії різних компонентів – REST, описаний Роєм Філдінгом [16]. Даний підхід до розробки накладає певні обмеження та правила на використання HTTP протоколу, завдяки чому підвищується продуктивність розробки та роботи системи, загальна архітектура комунікації у веб-мережі стає зрозумілішою, легшою для підтримування та розширювання. Підхід містить шість архітектурних правил-обмежень, п'ять з яких є обов'язковими. Якщо система повністю дотримується правил REST, її називають RESTful. Розробка моделі системи з врахуванням всіх правил REST займе багато часу, а тому виділимо та проаналізуємо основні правила, яких варто дотримуватись щоб робота системи стала зрозумілішою та продуктивнішою: клієнт-сервер, відсутність стану (stateless), однорідний інтерфейс, шарування компонентів системи. Розглянемо вказані правила.

Правило клієнт-сервер (не плутати з дворівневою архітектурою) встановлює чіткі рамки відповідальності для кожного компоненту: клієнт повинен відповідати лише за коректне відображення інформації в той час як

сервер повинен відповідати лише за збереження та зміну даних. Таким чином система розділена на компоненти, які можна розробляти незалежно один від одного, завдяки чому зменшується кількість можливих помилок.

Правило відсутності стану (stateless) вказує, що сервер не зберігає в собі жодної інформації, яка була б необхідна для обробки наступних запитів користувача. Тобто, на сервері не повинна зберігатись інформація про теперішній стан клієнта чи його сесії. Завдяки цьому, взаємодія між сервером та клієнтом стає зрозумілішою, за відсутності стану клієнта легше знайти наявні помилки в системі. Враховуючи вказані обмеження, клієнт з кожним запитом на сервер повинен надсилати інформацію, яка буде визначати стан клієнта та його сесію, наприклад токени автентифікації та авторизації, інформація про те, яку сторінку потрібно показати користувачу тощо.

Правило шарування компонентів системи визначає, що вся система повинна бути поділена на шари, де кожна частина/компонент одного шару може спілкуватись лише з такими самими компонентами вищого або нижчого шару. Таким чином компоненти системи мають низьку зв’язність між один з одним. Клієнт не повинен для кожного запиту визначати компонент серверу, до якого йому необхідно звернутись – він звертається лише до першого шару сервера, не знаючи скільки ще шарів та в якій послідовності задіяні для обробки запиту. Таким чином реалізація клієнта не залежить від реалізації серверу, яка для нього є чорним ящиком.

Однорідний інтерфейс означає те, що між клієнтом, сервером та різними компонентами сервера повинен бути узгоджений єдиний інтерфейс: кожен запит повинен точно визначати ресурс, який йому необхідний, відповідь повинна приходити в єдиному форматі. Кожен запит повинен містити всю необхідну інформацію для обробки цього повідомлення. Для оновлення ресурсу на сервері, запит повинен містити всі поля для оновлення.

Дотримання вказаного архітектурного стилю дозволить створити систему, яку буде легко оновлювати та масштабувати, змінювати без вимикання системи.

2.2 Аналіз, вибір та проектування архітектури серверної частини ПЗ

Якщо мова йде про клієнт-серверну архітектуру, можна виділити дві актуальні архітектури для серверної частини системи:

– мікросервісна архітектура;

– монолітна архітектура.

Розглянемо вказані архітектури, щоб зрозуміти їхні відмінності, переваги та недоліки та проведемо аналіз щоб зрозуміти, яку архітектуру варто обрати для розробки нашої системи.

Монолітна архітектура – архітектура, при якій вся система являє собою єдину програму, яка відповідає за весь необхідний функціонал, такий як збереження даних, їх обробка та відображення користувачу. Даний тип архітектури є загальнопоширеною практикою при побудові веб-додатків. Модель монолітної архітектури зображена на рисунку 2.2.

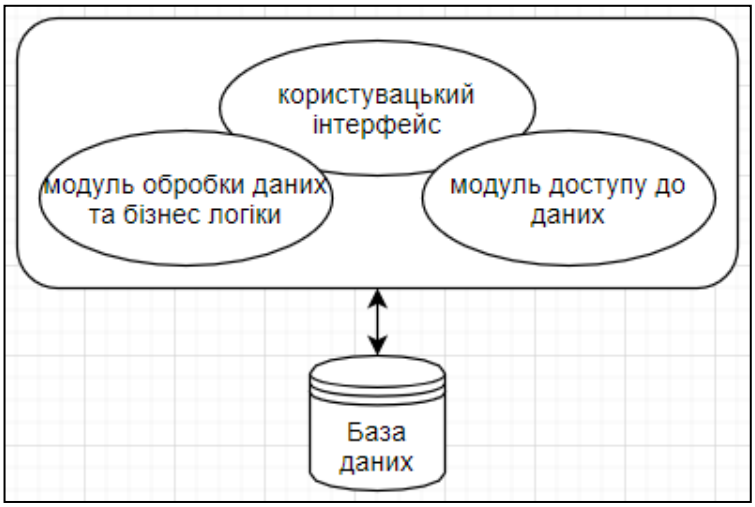


Рисунок 2.2 – Модель монолітної архітектури

Головною перевагою монолітної архітектури є те, що таку систему легше розробляти. Оскільки збереження даних, їх обробка та відображення знаходяться в одній програмі, таку систему легше створити, оскільки в ній відсутні окремі компоненти, які необхідно зв’язувати між собою. Завдяки цьому, розробка за

монолітною архітектурою займає менше часу. Також, середовища для розробки (IDE) проектувалися для розробки ПЗ як одного цілого, завдяки чому розробник використовує інструменти з максимальною ефективністю. Якщо система розроблена як єдине ціле – тестування такої системи є більш простою задачею, оскільки одна програма відповідає за всю логіку. Таку систему легше та швидше запускати, оскільки вона складається лише з однієї програми.

Тим не менш, такий підхід до розробки має велику кількість недоліків. Із розвитком проекту та збільшенням його функціоналу (відповідно й коду), між різними компонентами системи можуть виникати неявні залежності, які з часом уповільнять швидкість розробки, оскільки кожна зміна коду буде стосуватись більшої кількості коду та функціоналу. Між компонентами, які мають різну відповідальність розмиваються кордони, що може привести до проблем та багів, які важко вирішуються. Для вирішення проблеми зв'язності коду, його періодично необхідно переписувати (refactoring), що не завжди є можливим, оскільки замовник та клієнти системи іноді не розуміють всієї важливості чистого коду. Розуміння архітектури з розростанням системи стає все важчим, оскільки програма є єдиним цілим, як наслідок – в проект важче заходити новим розробникам. Внесення будь-якого нового функціоналу в проект вимагає повторної компіляції, яка з кожним разом займає більше часу. Якщо в програмі виникає певна помилка, така як витік пам'яті, вся система буде зупинена і клієнти не будуть в змозі виконувати хоч якісь дії. Вказану архітектуру варто застосовувати, якщо система не є великою або на розробку виділено обмаль часу.

Мікросервісна архітектура – архітектура, при якій розроблювана система розділяється на сервіси (програми), максимально невеликих, наскільки це можливо, розмірів [15]. Кожен з таких модулів відповідає за свою визначену задачу, та спілкується з іншими сервісами для вирішення інших задач. Таким чином, кожен з сервісів є слабо зв'язаним з іншими, та у випадку необхідності може бути заміненим іншим сервісом. Зазвичай, спілкування між сервісами відбувається за допомогою запитів протоколу HTTP. Модель мікросервісної

архітектури зображена на рисунку 2.3 (3 окремі мікросервіси з 2 БД).

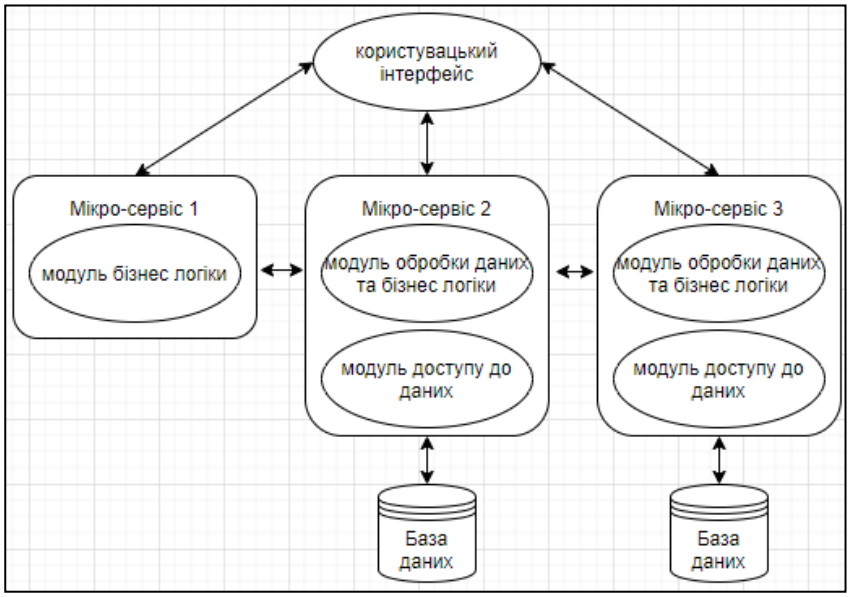


Рисунок 2.3 – Модель мікросервісної архітектури

Однією з головних переваг мікросервісної архітектури вважається те, що розробка окремих модулів може вестись незалежно різними розробниками. Кожен з таких сервісів може використовувати різні технології та мови програмування, і якщо використовувати HTTP як засіб спілкування – немає жодної різниці, як був реалізований той чи інший модуль. Зрозумілість коду при такій архітектурі не порушується, оскільки новий функціонал буде реалізований у новому мікросервісі. Новий учасник швидше ввійде в проект, оскільки його можна вивчати окремими сервісами, на відміну від ПЗ з монолітною архітектурою. Помилки в роботі програми легше локалізувати та виправити, оскільки така проблема буде стосуватись окремого компоненту системи. Якщо у випадку поломки програми-моноліту вся система перестане функціонувати, то при використанні даної архітектури перестане працювати лише частина функціоналу, і користувач зможе використовувати функції ПЗ частково. При такій архітектурі легше інтегрувати в проект нові технології та підходи до розробки, оскільки старі інструменти будуть стосуватись лише окремих сервісів. При масштабуванні, необхідно запускати лише ті сервіси які вимагаються, на

відміну від моноліту, де необхідно запустити весь комплекс, що в результаті може негативно вплинути на розподіл та використання ресурсів. Остання перевага, яку варто вказати для цієї архітектури – кожен з мікросервісів можна повторно використовувати в інших проектах, оскільки кожен з них відповідає за свою окрему задачу. Наприклад, модуль авторизації та автентифікації користувача, сервіс надсилання листів на пошту можна використовувати в різних проектах, таких як інтернет-магазин, соціальна мережа тощо.

Вказаний розподіл системи на менші програми має певні недоліки. Якщо в монолітній архітектурі для того щоб протестувати всю систему необхідно запустити одну програму – в даному випадку необхідно ввімкнути всі програми, що може викликати певні труднощі, оскільки з часом кількість мікросервісів може вимагати більше ресурсів, ніж є в локального комп'ютера. Також ця причина додає складності при написанні автоматичних тестів, які будуть перевіряти роботу всього функціоналу разом. При розробці таких систем необхідно враховувати те, що між окремими компонентам в ході виконання програми можуть виникати затримки (наприклад, через погане з'єднанням з інтернетом), які можуть спричинити проблеми, якщо певні дані в певному компоненті за планом мали з'явитись раніше, ніж вийшло насправді. Стежити за станом системи стає важче, оскільки в ній з'являється більша кількість програм, кожна з яких може завершити роботу помилкою. Запуск всього проекту повинен бути продуманий на перед – певні компоненти після старту можуть очікувати, що певні інші модулі вже є запущеними, і можуть бути використані ними. Для архітектури таких систем необхідно виділяти більше часу, особливо для розподілення відповідальності між сервісами. Якщо цього не робити, між ними можуть виникнути залежності, що може привести до того, що сервіси перетворяться в групи сервісів, які необхідно буде запускати разом як єдине ціле. Систему з даних груп сервісів можна назвати розділеним монолітом.

З розглянутих переваг та недоліків вище наведених архітектурних шаблонів, було вирішено обрати мікросервісну архітектуру, оскільки при такому

підході розробляються окремі компоненти, які потім можна буде повторно використати в інших проектах. Згідно з проаналізованою предметною областю, ми одразу можемо виділити дві задачі, які є незалежними одна від одної: комунікація з веб API Telegram та комунікація з блокчейном TRON. Дві дані задачі будуть винесені в окремі мікросервіси, які будуть спілкуватись між собою. Назвемо ці компоненти be-bot та be-tron відповідно (префікс be розшифровується як back-end, тобто сервіс серверної частини проекту). Сервіс be-bot відповідає за обробку повідомлень користувача, збереження його сесії, та весь процес спілкування з Telegram. Сервіс be-tron відповідає за надсилання нових переказів у блокчейн, витяг з нього історії транзакцій, нових переказів користувачам та інформації про баланси користувачів – узагальнюючи, обмін інформацією з блокчейном. Однак, в нас є задачі які вимагають роботи зі збереженими даними користувачів: для того щоб користувач міг створювати перекази, нам необхідно зберігати приватний ключ від його акаунту та пароль на випадок коли користувач вирішить експортувати гаманець, а також інформацію про оброблені транзакції блокчейну, щоб знати які транзакції-перекази користувачам є новими, і їх варто про це сповістити. Оскільки дана інформація не відноситься повністю до логіки роботи з Telegram або Tron, виокремимо її в окремий сервіс, який буде відповідати за зберігання та роботу з вказаними даними – be-core (core – від англ. ядро, тобто ядро системи). Також, сервіс be-bot буде мати власну базу даних для збереження сесій користувачів.

Розглянемо взаємодію вказаних сервісів при виконанні основного функціоналу, вказаного в постановці задачі – створення акаунту (як нового, так і імпорт існуючого), переказ коштів, повідомлення про нові вхідні перекази на акаунт користувачів, експорт гаманця, перегляд балансів та переказів в блокчейні. Для цього буде зручно використати діаграми послідовності, які показують в якій послідовності об’єкти обмінюються повідомленнями. Зазвичай, дані діаграми використовуються для відображення обміну інформацією між класами програми з об’єктно-орієнтованим підходом, але при архітектурному

рівні абстракції, дані діаграми чудово застосовуються для мікросервісної архітектури: кожен сервіс виступає в ролі класу, кожен запит є повідомленням від одного сервісу до іншого.

На рисунку 2.4 зображена діаграма послідовності створення нового акаунта (гаманця) в системі.

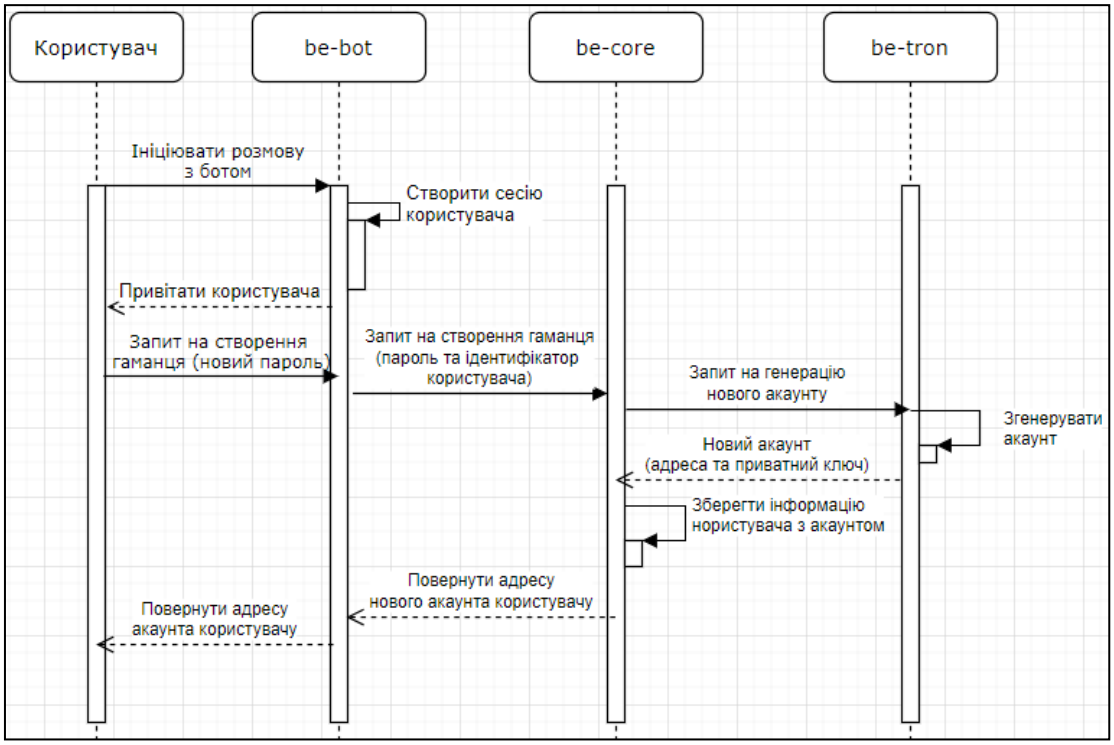


Рисунок 2.4 – Діаграма послідовності створення нового акаунта

Коли користувач ініціював розмову з ботом, бот створює нову сесію для користувача та вітає його у відповідь, з пропозицією створити акаунт. Коли користувач створив пароль для гаманця та натиснув кнопку “створити”, be-bot звертається до be-core з метою створити новий гаманець. Be-core звертається до be-tron задля генерації нового акаунту. Be-tron повертає новий акаунт сервісу be-core, після чого останній зберігає інформацію про користувача з його новим акаунтом. Після цього, be-core дає відповідь сервісу be-bot з адресом акаунта, останній повідомляє користувача про успішне створення гаманця з його адресом. Імпорт гаманця відбувається аналогічним чином, з єдиною різницею – користувач окрім створення паролю для акаунту вводить приватний ключ

власного гаманця, з якого отримується адреса сервісом be-tron під час виконання створення акаунта.

На рисунку 2.5 зображена діаграма послідовності переказу коштів.

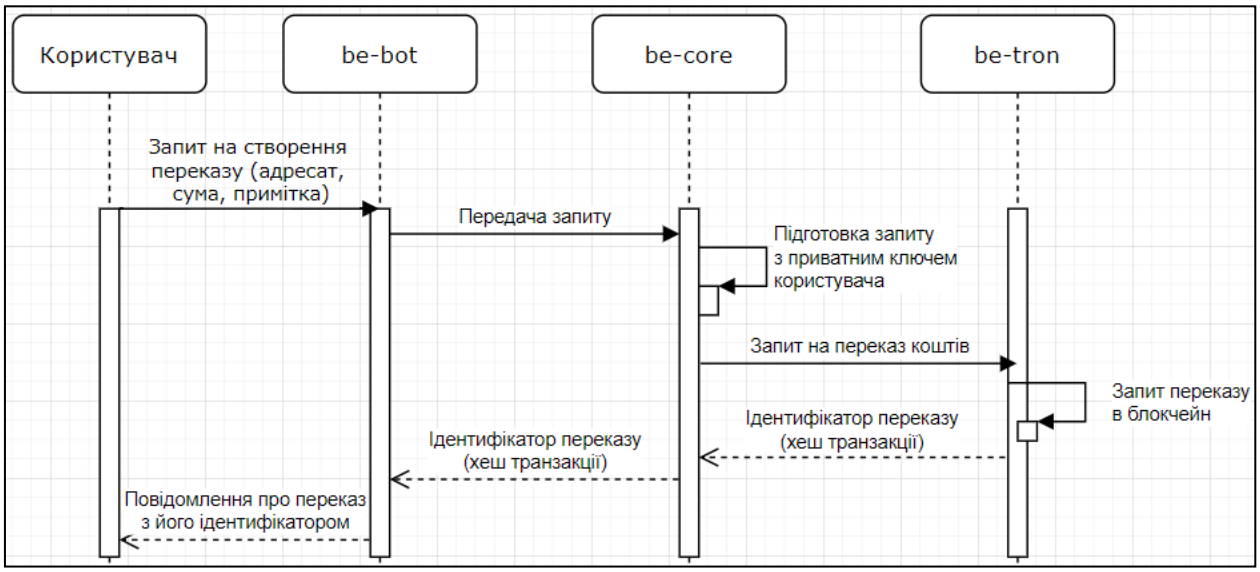


Рисунок 2.5 – Діаграма послідовності переказу коштів

При переказі коштів, користувач повинен ввести адресата та суму переказу, примітку до переказу (необов'язковий параметр). Be-bot передає цю інформацію сервісу be-core, де відбувається формування запиту до be-tron. Be-core бере приватний ключ акаунта користувача в базі даних, надсилає його з запитом до be-tron. Be-tron формує та надсилає запит до блокчейну, отримуючи у відповідь ідентифікатор переказу (хеш транзакції), який повертається користувачу проходячи попередні сервіси. Таким самим чином працює застава коштів та вивід TRX з під застави.

Коли користувачу необхідно переглянути певні баланси або інформацію про певні перекази – сервіс be-bot буде напряму звертатись до сервісу be-tron з відповідними параметрами – адресом акаунту або хешем транзації, після чого інформація буде формуватися та відображатися користувачу.

На рисунку 2.6 зображена діаграма послідовності експорту гаманця.

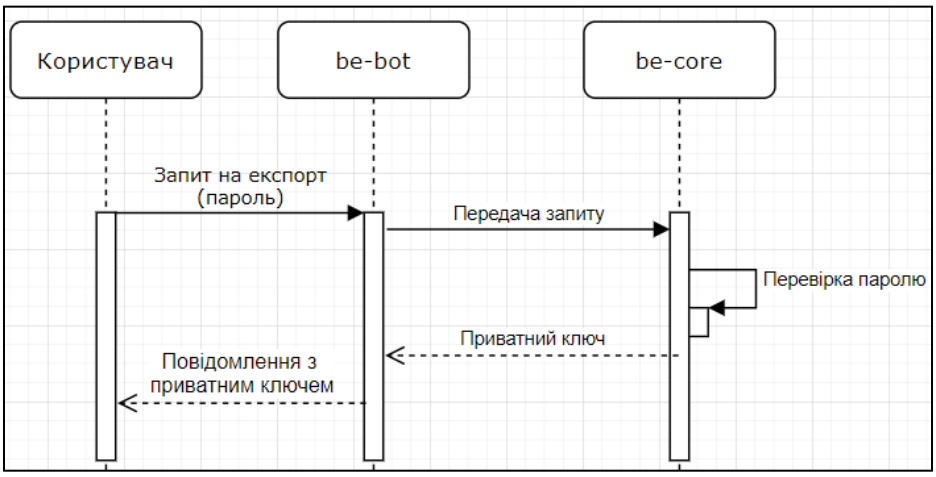


Рисунок 2.6 – Діаграма послідовності експорту гаманця

Для експорту гаманця, користувач вводить свій пароль. Даний запит з паролем сервіс be-bot надсилає сервісу be-core, де останній перевіряє введений пароль зі збереженим при створенні гаманця. Якщо пароль вірний, користувачу віддається його приватний ключ. В іншому випадку, користувач отримає повідомлення про невірний пароль. Ідентичним чином працює видалення акаунту з бота, якщо введений пароль вірний, з бази даних be-core видаляє запис про гаманець користувача з паролем.

На рисунку 2.7 зображена діаграма послідовності сповіщень про нові вхідні перекази користувачам бота.

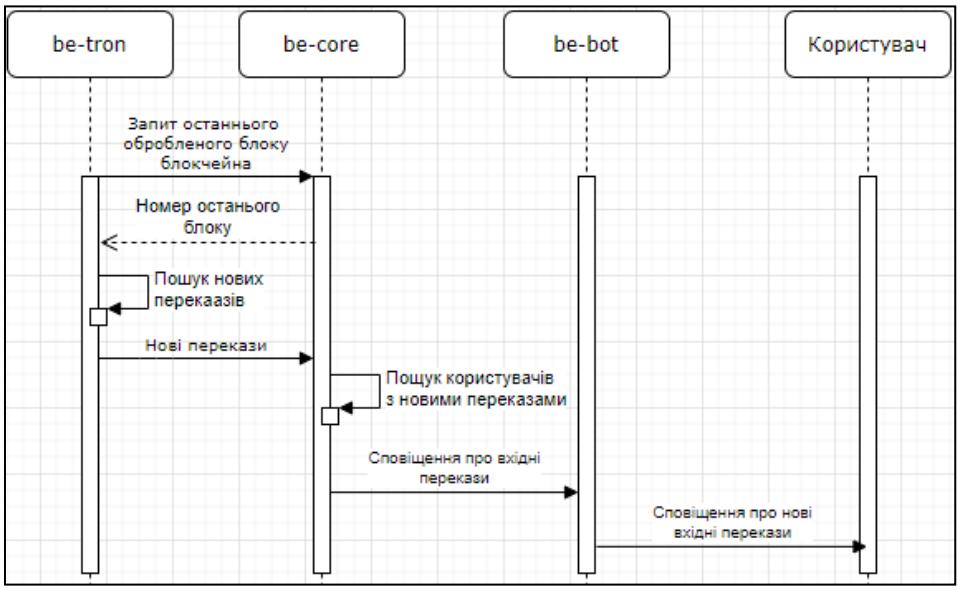


Рисунок 2.7 – Діаграма послідовності сповіщень про нові вхідні перекази

Для пошуку нових вхідних переказів, сервіс be-tron робить запит до be-core з метою отримати останній оброблений блок транзакцій блокчейну – це і буде блоком, від якого варто сканувати блокчейн на наявність нових вхідних транзакцій. Після сканування блокчейну, be-tron надсилає всі перекази сервісу be-core, який в свою чергу фільтрує серед цих переказів ті, які є вхідними для користувачів бота. Be-core оновлює інформацію про останній опрацьований блок сервісом be-tron для його наступного запиту, після чого формує сповіщення користувачам про нові вхідні кошти. Дані сповіщення надсилаються до be-bot, який і розсилає їх потрібним користувачам.

Після проведеного проектування взаємодії мікросервісів, можемо зобразити їх модель (рисунок 2.8).

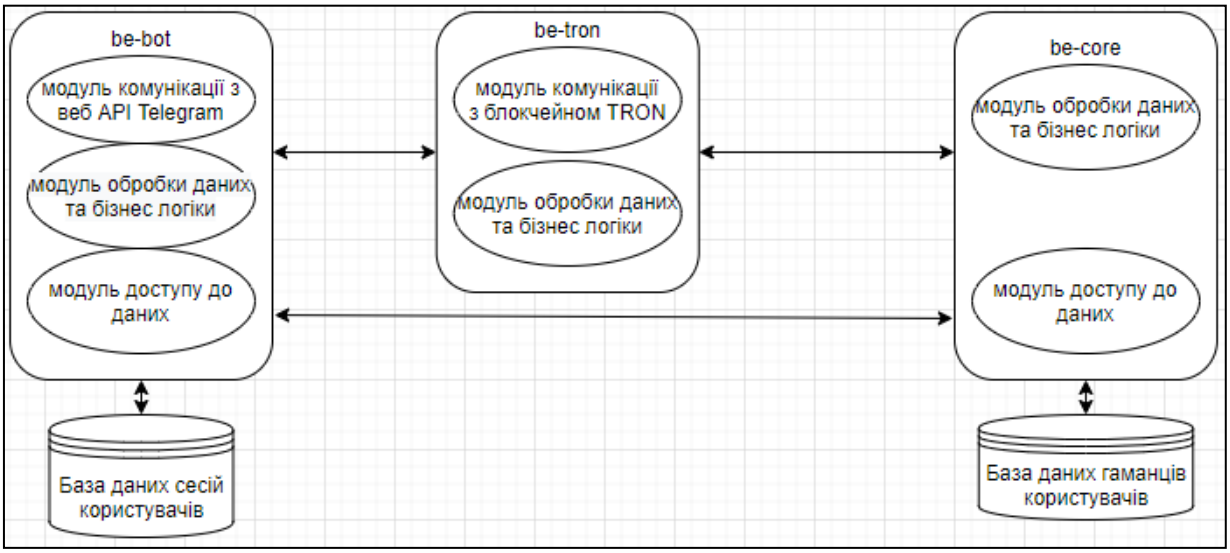


Рисунок 2.8 – Модель мікросервісів розроблюваної системи

2.3 Аналіз та вибір типу бази даних, проектування структури бази даних

Для вибору бази даних (БД), проаналізуємо існуючі типи баз даних, щоб зрозуміти їх відмінності. Основними типами БД (за моделлю зберігання та організації даних), які активно використовуються при розробці ПЗ є реляційні та нереляційні бази даних.

Реляційні бази даних побудовані на реляційній моделі даних – такі БД зберігають дані в двовимірних таблицях, які мають певну кількість стовпців (атрибутів) для різних властивостей даних які зберігаються. Кожен стовпчик таблиці має власну назву та зберігає лише один тип даних – наприклад число, стрічка, булева зміна тощо. Кожен рядок (кортеж) в таблиці являє собою окремий запис, одиницю даних, а всі елементи в одному стовпці містять лише однорідні дані одного типу. Порядок рядків в таблиці не вказує їх послідовності – її варто визначати власною логікою сортування. В таблиці не може бути однакових записів. Кожна таблиця повинна містити стовпець або групу стовпців, які унікально ідентифікують кожен запис – такі стовпці називаються первинним ключем. Відповідно, в таблиці не може існувати два записи з однаковим значенням такого стовпця – порушиться ідентифікація даних. Для зв'язку між таблицями використовуються зовнішні ключі – стовпці в одній таблиці, які відповідають первинному ключу іншої таблиці. Наприклад, таблиця "замовлення" може містити поле "ідентифікатор користувача", значення якого відповідає первинному ключу таблиці "клієнт". Основною перевагою реляційних БД є наявність підтримки транзакцій – виконання декількох операцій як одного цілого. Тобто, в БД відбудуться зміни від всіх операцій разом, або змін не відбудеться взагалі. Наявність підтримки транзакцій в базі даних забезпечується чотирма властивостями, які мають бути притаманні базі даних:

- атомарність (Atomicity) – транзакція не буде виконана не повністю;
- узгодженість (Consistency) – до та після транзакції жодні дані в системі не мають суперечливого стану;
- ізольованість (Isolation) – жодні зміни під час виконання транзакції не будуть доступними нікому, допоки транзакція не завершена;
- довговічність (Durability) – БД гарантує, що результати транзакцій будуть збережені навіть після збоїв системи.

Дані чотири властивості скорочено називаються ACID, і вони є важливими для системи, в якій транзакції є обов'язковою умовою – наприклад банки, біржі

активів, інші системи які працюють з коштами. Найпопулярніші реляційні бази даних – MySQL, Oracle, Postgres.

Нереляційні бази даних, які ще називають NoSQL – БД, які не використовують реляційні відношення. Часто такі бази є орієнтованими на документи, тобто кожен запис в базі даних – це документ з потрібною інформацією. Сукупність однакових за структурою документів називають колекцією (аналог таблиці). Такі БД даних є зручнішими та гнучкішими в використанні за реляційні – відсутня необхідність проектування таблиць з визначенням типів даних, замість цього БД просто зберігає вказаний документ. Також, даний тип баз даних швидше працює за реляційні бази даних. Недоліком таких баз даних є те, що в них відсутній повноцінний механізм транзакцій – певні бази даних гарантують консистентні зміни в межах лише одного документа. Найпопулярніші приклади нереляційних баз даних – MongoDB, CouchDB.

Проаналізувавши переваги та недоліки вказаних типів БД було прийняте рішення обрати нереляційні бази даних, оскільки для них не потрібно детально проектувати структури даних, таблиці та їх зв’язки.

В системі будуть існувати дві окремі БД – для сервісу be-core, та для сервісу be-bot. БД в be-core зберігає два види документів – інформація про останній оброблений блок блокчейну TRON та інформацію про гаманець користувача. Назвемо відповідні колекції Tron Block та User. У базі буде міститись лише один документ в колекції Tron Block, він міститиме наступні поля: last processed block – номер останнього опрацьованого блоку з блокчейну, update timestamp – час останнього оновлення документа. Документи колекції User міститимуть наступні поля: telegram user id – ідентифікатор користувача Telegram; chat id – ідентифікатор чату, в якому користувач спілкується з ботом ; active wallet address – адреса гаманця користувача ; active wallet password – пароль користувача від гаманця; active wallet private key – приватний ключ від гаманця; create timestamp – час створення документа; update timestamp – час оновлення документа. БД сервісу be-bot міститиме одну колекцію – User Session, яка відповідає за

збереження інформації про сесію користувача. Дані документи зберігатимуть велику кількість полів, які необхідні для функціонування бота – детальніша структура даного документа буде наведена на етапі реалізації ПЗ, де буде обґрунтована необхідність встановлених полів.

2.4 Проектування інтерфейсу користувача

Розглянемо приклад реального Telegram бота – сервіс для створення та використання тимчасової поштової скриньки “tmpmailbot” (рисунок 2.9).

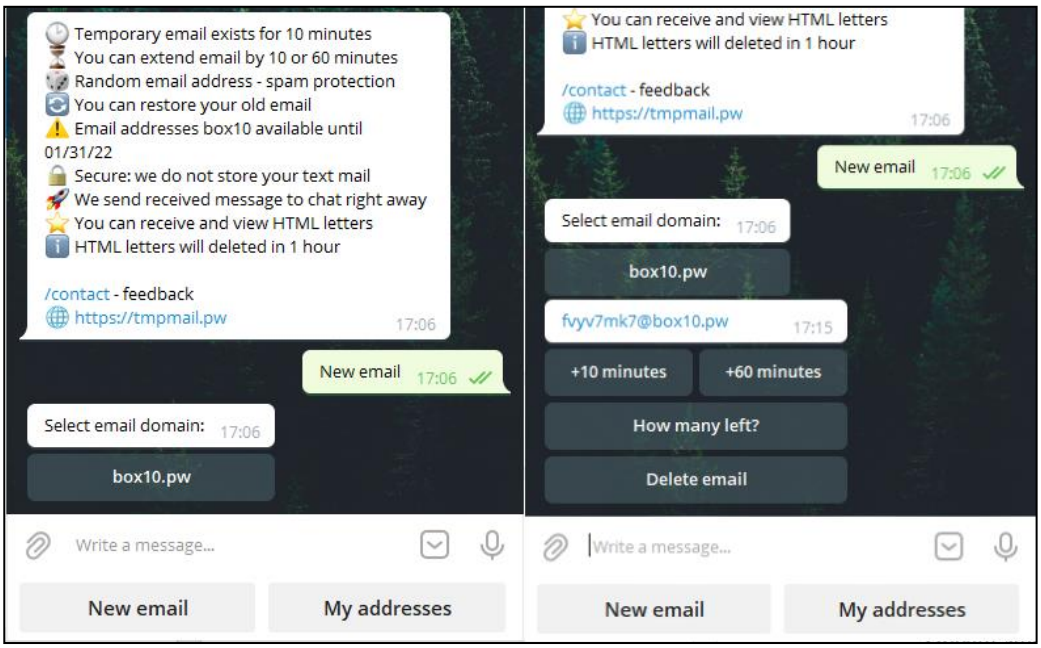


Рисунок 2.9 – Приклад Telegram-боту

Зображення ліворуч вказує відповідь на натиск кнопки “New email” знизу екрану, зображення праворуч показує відповідь на натиск кнопки “bot10.pw”, вбудовану в повідомлення: На даному рисунку, ми можемо виділити два основні елемента бота, з якими взаємодіє користувач: основна клавіатура знизу інтерфейсу, та клавіатура вбудована в повідомлення. Їх основна відмінність в тому, що основна клавіатура при дії користувача (натиск кнопки) надсилає від імені користувача повідомлення з тим текстом, який був написаний на кнопці,

коли вбудована клавіатура при такій дії надсилає боту інформацію про те, що користувач натиснув певну кнопку. Тобто, якщо користувач введе вручну таке саме повідомлення, яке міститься на кнопці основної клавіатури – бот буде опрацьовувати його, як натиск відповідної кнопки. З кнопками вбудованими в повідомлення користувач не зможе повторити такої дії, оскільки натиск такої кнопки при отриманні повідомлень від веб API Telegram виглядає іншим чином, а не звичайним текстовим повідомленням від користувача. Таким чином, ці два типи клавіатури умовно розділяють інтерфейс наступним чином: основна клавіатура виступає в ролі головного меню, яке може та має бути доступною під час любого стану програми; клавіатура вбудована в повідомлення являє собою меню, яке відкривається користувачу як перехід з головного меню до певного конкретного функціоналу програми, і може бути доступним лише тоді, коли користувач знаходиться саме в цьому розділі від головного меню. В нашій системі існує два стани головного меню: коли користувач має активний гаманець, та коли головний гаманець відсутній. Відповідно до діаграми варіантів використання визначеній в першому розділі, побудуємо 2 клавіатури основного меню, які будуть використовуватись для користувачів без та з активним гаманцем (рисунки 2.10 та 2.11 відповідно).

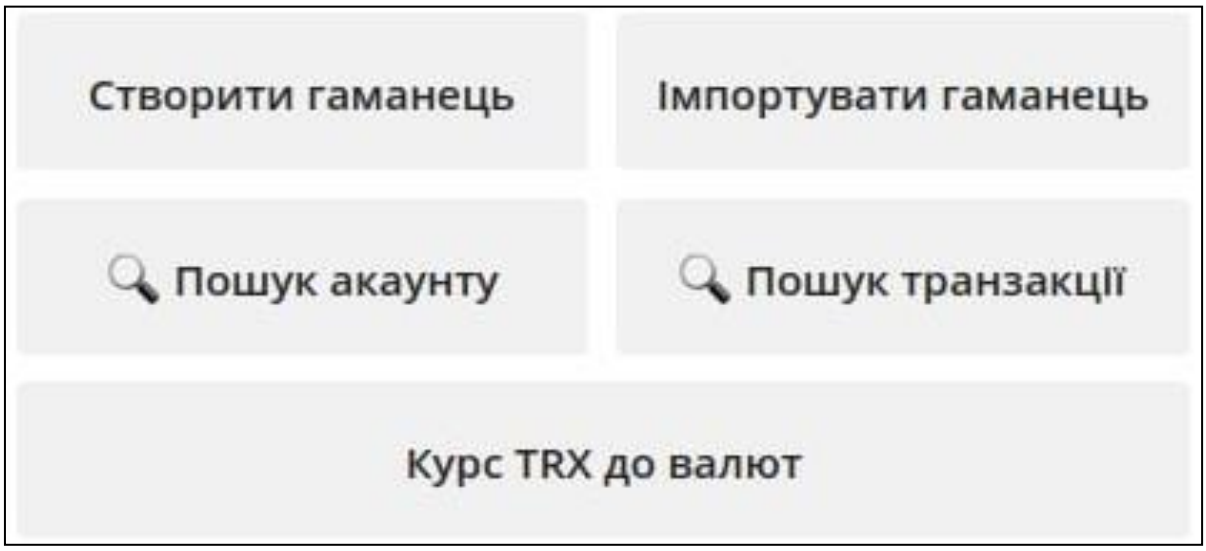


Рисунок 2.10 – Головне меню користувача без активного гаманця

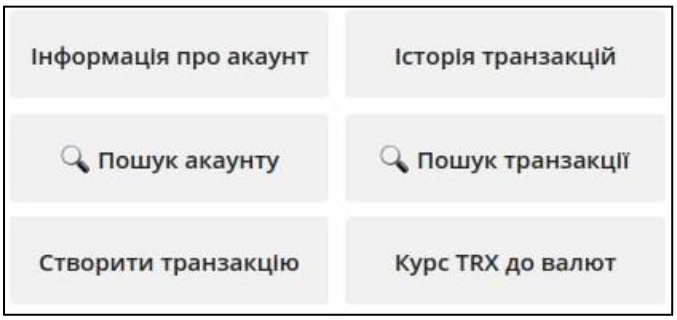


Рисунок 2.11 – Головне меню користувача без активного гаманця

Клавіатури вбудовані в повідомлення використовуватимуться, коли користувачу необхідно буде виконувати певні дії, які доступні як вкладки після переходу з головного меню – тобто, створення гаманця, переказ коштів та інші. Якщо користувачу необхідно заповнити декілька полів (наприклад, при переказі коштів) – клавіатура буде містити кнопку для заповнення кожного поля. Бот буде заповнювати певне поле лише тоді, коли користувач ввів його значення, попередньо натиснувши відповідну кнопку. На рисунках 2.11-2.13 зображено три стани вбудованої в повідомлення клавіатури бота для переказу коштів, коли користувач вирішив заповнити певну інформацію – головне меню переказу коштів (рисунок 2.12), введення адресата для переказу після натиснення кнопки “Адрес отримувача” (рисунок 2.13), меню після введення адреси (рисунок 2.14).

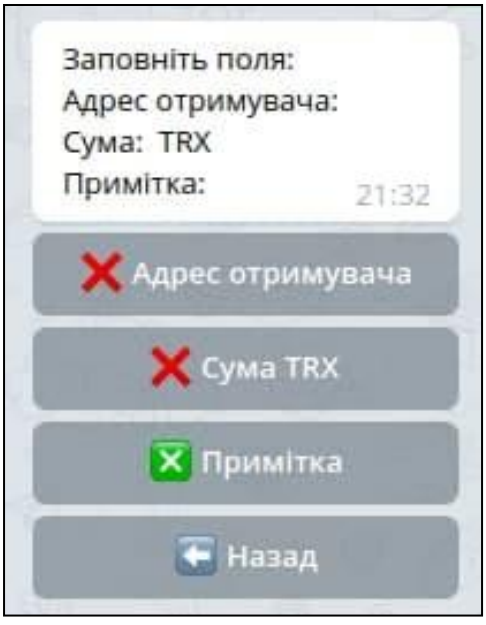


Рисунок 2.12 – Меню переказу коштів (незаповнене)

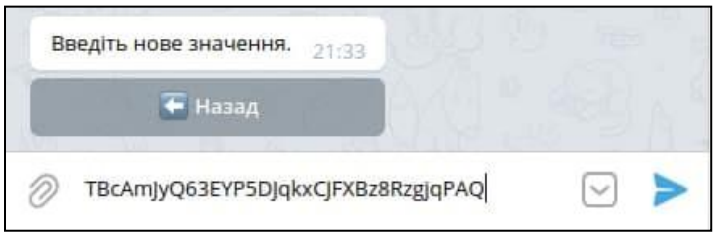


Рисунок 2.13 – Меню переказу коштів (заповнення адреси отримувача)

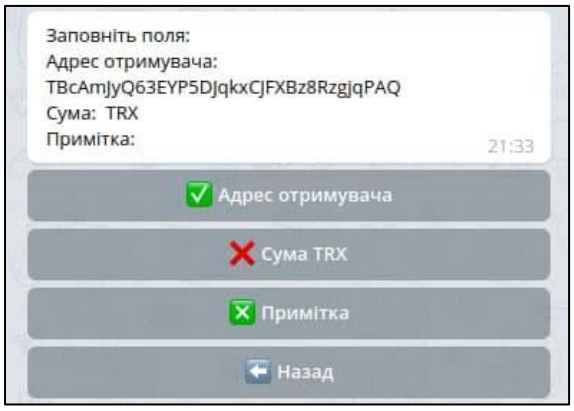


Рисунок 2.14 – Меню переказу коштів (адреса отримувача вказана)

На вбудованій клавіатурі, поля які є обов’язковими до заповнення, але ще не заповнені будемо позначати червоним хрестиком. Поля, які є не обов’язковими і не заповнені будемо позначати хрестиком в зеленому контурі. Заповнені поля будемо позначати зеленою галочкою. Інші дії з використанням вбудованих клавіатур будуть відбуватись аналогічним чином. Якщо якась кнопка виконує логіку без введення даних – вона просто переміщає користувача в потрібне меню. Користувачу відображаються лише ті кнопки, які він може використати – тобто якщо всі необхідні дані не заповнені – створити та відправити переказ неможливо.

2.5 Аналіз та вибір технологій і методів реалізації системи

Більшість мов програмування мають можливості для розробки back-end проекту. Враховуючи наявність великої кількості різних фреймворків та інструментів, об’єм інформації та документації, я обрав об’єктно-орієнтовану

мову програмування Java для реалізації системи. До переваг Java також можна віднести те, що її додатки запускаються на JVM (Java virtual machine), тобто розроблений додаток можна запустити як на Windows, так і MacOS, Linux та інших ОС.

Найбільш поширеним фреймворком для побудови back-end веб-додатків на Java є універсальний фреймворк Spring, який підтримує принципи інверсії контролю (Inversion of Control, IoC) та впровадження залежностей (Dependency Injection, DI). Інверсія контролю – принцип, за яким написаний нами код керується фреймворком, а не навпаки: у випадку використання якоїсь бібліотеки, ми як програмісти зобов'язані викликати класи цієї бібліотеки, щоб нею скористатись – при IoC, наш код викликається самим фреймворком. Таким чином розвинуті фреймворки з великою кількістю функціоналу зменшують кількість коду, який потрібно написати розробнику – необхідно реалізувати лише бізнес логіку, все інше за нас зробить Spring. Впровадження залежностей – шаблон, при якому програмні компоненти отримують свої залежності ззовні, не переймаючись логікою створення цих залежностей. Зазвичай, такі залежності передаються класу через конструктор, присвоєння поля класу або метод. Такий підхід зменшує зв'язність компонентів програми, а за допомогою IoC від Spring, програмісту не потрібно перейматись впровадженням залежностей – фреймворк самостійно виконає необхідні впровадження. Самостійно, Spring не надає можливостей створювати веб-сервіси – для цього необхідно використати модуль Spring Web, який містить в собі необхідні компоненти для побудови ПЗ за шаблоном Model-View-Controller (MVC). Даних компонентів досить, щоб створити веб-додаток у форматі WAR (web archive), який необхідно буде помістити під контроль веб-серверу. Щоб не виконувати вказаних маніпуляцій для кожного мікросервісу, використаємо інструмент Spring Boot – це надбудова над фреймворком Spring, яка автоматично налаштовує велику кількість конфігурацій Spring, Spring Web замість програміста. Великою перевагою використання Spring Boot для Spring Web є те, що нам не потрібно розміщувати

компільований код в веб-сервер – Spring Boot має вбудований сервер Tomcat, який використовується для цієї цілі. Для збереження паролів користувачів у захищеному вигляді – так званий хеш пароля, та перевірки введеного користувачем пароля з даним хешем використаємо можливості модуля Spring Security, який надає функціонал для захисту даних.

Розглянемо та проаналізуємо дві поширені нереляційні БД та виберемо одну з них для нашої системи – Apache CouchDB та MongoDB. Apache CouchDB – нереляційна документ-орієнтована БД, яка зберігає документи у форматі JSON. Особливістю бази даних вважається те, що вона містить в собі вбудований веб-сервер, який дозволяє використовувати її за допомогою HTTP-запитів. CouchDB націлена на стійкість до розділення на вузли та доступність даних – в такому випадку, узгодженість даних не гарантується. MongoDB – також нереляційна об’єктно орієнтована БД, яка зберігає дані в JSON форматі. Одна з головних переваг MongoDB – потужна мова запитів, яка дозволяє виконувати важкі операції, агрегації та маніпулювання над даними. На відміну від CouchDB, дана БД має в пріоритеті узгодженість даних та розділення на вузли, а тому її безперебійна доступність не гарантується. Ще однією перевагою цієї БД для нашої системи можна вважати те, що Spring надає модуль Spring Data MongoDB, який дозволяє зручно використовувати БД в коді. Враховуючи особливості наведених БД, була обрана MongoDB за рахунок її інтеграції з Spring.

Для менеджменту залежностей проекту, застосовують спеціальні системи автоматичного збирання, які виконують всю роботу з залежностями замість нас. Для Java зробимо вибір між найпоширенішими системи – Apache Maven та Gradle. Apache Maven є старішим проектом, де робота з залежностями описується в форматі XML. Gradle дає описувати залежності на предметно-орієнтованій мові Groovy, яку додатково можна використовувати для написання різноманітних Gradle-завдань. Для нашої системи різниці між вказаними інструментами не буде відчутно, але було вирішено використовувати Gradle, тому що в такому випадку необхідно буде виконувати менше налаштувань, ніж з Apache Maven.

У цьому розділі ми визначили, що наша система буде побудована за клієнт-серверною архітектурою. Було проведено аналіз серверних архітектурних підходів, та була обрана мікросервісна архітектура, враховуючи її переваги в зрозумілості компонентів коду перед монолітною. Були виділені 3 мікросервіси та визначена їх взаємодія при виконанні необхідного функціоналу. Був проведений аналіз типів баз даних та обрана нереляційна БД MongoDB у зв’язку з зручністю її використання, було визначено структуру сутностей для БД. Ми спроектували користувацький інтерфейс, визначивши поняття головного та вкладеного меню у вигляді відповідних клавіатур. Було вирішено використовувати мову програмування Java з фреймворком Spring та його модулями, а також систему автоматичного збирання Gradle.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування мікросервісів

Для всіх спроектованих мікросервісів, можна виділити наступні загальні модулі, які відповідають за виконання визначеного функціоналу в програмі (розуміється, кожен мікросервіс буде мати власну реалізацію цих модулів): web, service, exception, integration, configuration, constants, database. Модуль exception містить класи, які відносяться до логіки обробки помилок при виконанні функціоналу програми – реалізація та приклад використання будуть наведені в реалізації. Модуль constants містить класи типу перечислень (enum), які використовуються в різних класах, які призначені для збереження даних. Приклад констант – тип транзакції, статус транзакції та інші. Модуль database відповідає за класи, які реалізують собою роботу з БД. Даний модуль містить два пакети класів – repository та model, які відповідно реалізують собою репозиторії (класи, використовуючи які програма звертається до БД), та моделі – класи, які є репрезентаціями документів з БД. Модуль integration відповідає за комунікацію з іншими мікросервісами – він містить класи які безпосередньо виконують комунікацію а також класи для передачі даних (DTO класи), об’єкти яких призначені для серіалізації та десереалізації у JSON формат, який використовується при спілкуванні між мікросервісами. Модуль configuration відповідає за початкове налаштування мікросервісу – наприклад, вказує класам для комунікації з іншими мікросервісами куди їм варто звертатись, налаштовує часовий пояс мікросервісу тощо. Модуль web містить класи-контролери, які відповідають за обробку вхідних HTTP запитів від інших мікросервісів, та класи-DTO для передачі даних. Модуль service відповідає за обробку запитів, які приймає модуль web.

Проведемо детальне проектування мікросервісу be-core. Даний сервіс складається зі всіх базових модулів, та модуля util. Модуль util містить в собі класи-помічники – в даному випадку там міститься клас CryptoAES, який

відповідає за шифрування та розшифрування даних, він використовується для шифрування приватного ключа користувача. Модуль integration містить два пакети – для інтеграції з мікросервісами be-tron та be-bot. Модуль database містить два класи-репозиторії – TronBlockRepository та UserRepository, які відповідають за доступ до колекцій БД TronBlock та User відповідно. Перша колекція зберігатиме останній оброблений блок сервісом be-tron при пошуку нових вхідних переказів. Також, даний модуль містить класи-моделі для даних колекцій. Модуль web містить два класи-контролери – UserController та TransactionController. Перший клас відповідає за обробку наступних HTTP запитів, що стосуються користувача – створення нового гаманця, експорт та видалення гаманця. Другий клас відповідає за запити, пов'язані з транзакціями в блокчейні – запити на створення переказу, застави коштів та відміни застави, а також обчислення вартості вказаних транзакцій. Приклади DTO класів будуть наведені в реалізації. Модуль service, відповідно до пакету web, має два класи-сервіси, які обробляють HTTP запити, які прийняли класи-контролери. Основна задача класу TransactionController та його сервісу – підготувати та надіслати запит до be-tron, який виконуватиме запити до блокчейну.

Проведемо детальне проектування мікросервісу be-tron. Даний сервіс складається зі всіх базових модулів (окрім database), та модуля util. Модуль util містить клас Calculator, який використовується для точних обчислень. Модуль integration містить два пакети – для інтеграції з мікросервісом be-core та веб-сервісом TronScan, який був створений розробниками блокчейну TRON та є зручним засобом для перегляду блокчейну (акаунтів, блоків та транзакцій тощо). Даний веб-сервіс надає зручний веб API для отримання інформації з блокчейну. Модуль web містить два класи-контролери – AccountController та TransactionController. Перший клас відповідає за обробку наступних HTTP запитів, що стосуються акаунтів блокчейну – створення нового акаунту, отримання адреси акаунту з приватного ключа (імпорт гаманця), а також отримання ресурсів певних гаманців. Другий клас відповідає за обробку запитів

від be-core, пов’язаних з транзакціями. Модуль service виконує обробку запитів, які прийняли контролери. Для роботи з блокчейном, такої як надсилання нових транзакцій, генерація нових акаунтів та інших операцій, які не будуть виконуватись за допомогою API TronScan, використаємо open-source бібліотеку-SDK trident, яка активно розробляється спільнотою блокчейну. Окремо виділимо клас NewIncomeTransactionProcessor, який буде з певною періодичністю сканувати нові блоки в блокчейні на наявність вхідних переказів користувачам бота, та передавати їх мікросервісу be-core.

Мікросервіс be-bot буде відрізнятись від попередніх модулів: даний модуль повинен містити функціонал створення повідомлень у Telegram для користувача, які мають містити правильно сформовані повідомлення та клавіатури інтерфейсу. Для комунікації з веб API Telegram була використана бібліотеку open-source TelegramBots, яка надає функціонал отримання та відправки повідомлень користувачам у вигляді зручних класів.

Основним класом для обробки та надсилання повідомлень користувачам є BotGateway. Даний клас містить функціонал прийому повідомлень від користувачів, та функціонал по відправці повідомлень у відповідь. Коли приходить повідомлення від користувача, BotGateway використовує клас SessionService, який відповідає за створення, збереження та оновлення сесії користувача, а також її заповнення інформацією, яку користувач написав в повідомленні(або яку кнопку він натиснув). Після отримання сесії (клас UserSession) від SessionService, BotGateway передає об’єкт сесії користувача та об’єкт повідомлення від користувача класу UserUpdateProcessor, який відповідає за обробку повідомлення від користувача. UserActionProcessor на основі введених даних та теперішнього стану сесії користувача визначає, яке вікно інтерфейсу необхідно відобразити наступним (з яким текстом та клавіатурою відправити повідомлення користувачу у відповідь). Опис тексту (і кнопок для клавіатури) для кожного окремого повідомлення-вікна інтерфейсу міститься у класі MessageConfigurationStorage. Для даних конфігурацій був створений

спеціальний клас MessageConfiguration, який містить в собі налаштування для кожного повідомлення окремо. Після вибору потрібного об’єкта-конфігурації інтерфейсу, клас UserUpdateProcessor оперує наступними класами: UserActionProcessor, MessageKeyboardProcessor, MessageTextProcessor та MessageSender. UserActionProcessor відповідає за обробку дії користувача – якщо користувач ввів одне з полів для переказу коштів, даний клас відповідає за перевірку введеного значення та його збереження в сесії. Якщо користувач натиснув кнопку для відправки переказу коштів – задача цього класу відправити ці данні мікросервісу be-core та підготувати його відповідь для відображення користувачу. MessageKeyboardProcessor відповідає за налаштування клавіатури для повідомлення-відповіді користувачу: з об’єкту класу MessageConfiguration обираються потрібні кнопки (формується їх текст та функціонал), та зберігаються в сесію користувача, щоб на наступну дію від користувача, бот знав що саме хоче користувач від бота. MessageTextProcessor відповідає за підготовку тексту повідомлення-відповіді на основі конфігурацій тексту в об’єкті-конфігурації класу MessageConfiguration. MessageSender відповідає за надсилання повідомлення користувачу з створеною клавіатурою та текстом повідомлення. Мікросервіс be-bot складається зі всіх загальних модулів, як і інші мікросервіси. Модуль web містить клас DepositNotificationController, який відповідає за прийом вхідних сповіщень для користувачів. Він передає сповіщення класу-сервісу DepositNotificationService, який за допомогою клсу MessageSender надсилає їх користувачам. Модуль integration містить класи для комунікації з be-core, be-tron та сторонніми біржами криптовалют (Binance та Cryptex24) для отримання актуальних курсів TRX до долара та гривні. Наведені біржі надають для розробників API, який можна використати для отримання актуальної інформації про ціни криптовалют. Модуль database відповідає за збереження сесій користувачів. Інші загальні модулі містять аналогічні класи, описані для попередніх мікросервісів. Діаграми класів для побудованих мікросервісів, які відображають структуру програм подані на рисунках Б.1 – Б.11 (додаток Б).

3.2 Розробка програмних модулів

При розробці коду для зменшення його об’єму була використана бібліотека Lombok, яка замінює типові методи для класів анотаціями. Наприклад, анотація @Getter над класом згенерує для всіх приватних полів get-методи, @Builder згенерує для класу реалізацію шаблону будівельник (builder pattern) тощо. Кожен мікросервіс містить наступні обов’язкові елементи: файл властивостей, файл залежностей, запуск фреймворку Spring. Файл властивостей називається application.properties, та містить різні властивості проекту у вигляді ключ-значення, наприклад порт який буде прослуховувати даний мікросервіс, HTTP компоненти URI ресурсів інших мікросервісів, секретні ключі для шифрування даних (використовується у be-core), ключі доступу до телеграм-боту (використовується у be-bot) та інші. Файл залежностей build.gradle містить налаштування залежностей для мікросервісу. Кожен мікросервіс ініціює роботу фреймворку Spring викликом наступного коду в методі main (зображено фрагмент коду з мікросервісу be-core):

```
@SpringBootApplication
public class CoreApplication {
        public static void main(String[] args) {
                SpringApplication.run(CoreApplication.class, args);
        }}
```

Розглянемо деталі реалізації основних модулів на прикладі мікросервісу be-core. Розпочнемо з модуля exception, який у всіх мікросервісах має ідентичну реалізацію. Якщо під час виконання HTTP запиту від іншого сервісу трапилась певна помилка – даний мікросервіс повинен віддати у відповідь інформацію, яка ідентифікує помилку, щоб інший мікросервіс міг зрозуміти, що йому варто робити в такій ситуації. Клас ErrorInfo є DTO класом, який містить інформацію про повернену помилку:

```
package edu.diploma.ms.bot.business.dto;
import lombok.*;
```

```
import java.util.List;
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ErrorInfo {
    private List<String> messages;
    private Integer code;
}
```

Значення code – унікальний ідентифікатор помилки між мікросервісами, список messages містить текстовий опис помилки. Всі створені нами класи-помилки розширюють клас BaseRuntimeException:

```
package edu.diploma.ms.bot.business.exception;
import lombok.Getter;
@Getter
public class BaseRuntimeException extends RuntimeException {
    private final ExceptionCode exceptionCode;
    BaseRuntimeException(ExceptionCode exceptionCode, String message, Throwable
throwable) {
        super(message, throwable); this.exceptionCode = exceptionCode;
}

    BaseRuntimeException(ExceptionCode exceptionCode, String message) {
        super(message); this.exceptionCode = exceptionCode;
    }
    ExceptionCode getExceptionCode() {
        return exceptionCode;
    }
    @Override
    public String getMessage() {
        return super.getMessage();
    }}
```

Даний клас розширює стандартний клас мови Java RuntimeException, та містить поле типу ExceptionCode – клас, який містить опис для помилок цього мікросервісу та опис інших помилок з інших мікросервісів:

```
package edu.diploma.ms.bot.business.exception;
import lombok.*;
import org.springframework.http.HttpStatus;
@Getter
@RequiredArgsConstructor
public enum ExceptionCode {
    INVALID_PARAMETERS(4000102,"Invalid parameters.",HttpStatus.BAD_REQUEST),
    WRONG_PASSWORD(4000202, "Wrong password", HttpStatus.BAD_REQUEST),
    INTERNAL_SERVICE(5000102,"Internal service issue",
HttpStatus.INTERNAL_SERVER_ERROR),
    UNKNOWN (9999902, "Unknown error", HttpStatus.INTERNAL_SERVER_ERROR),
```

```
    private final int errorCode;
    private final String message;
    private final HttpStatus status;

}
```

Поля errorCode (ідентифікатор помилки) та message (текстовий опис помилки) використовуються при формуванні ErrorInfo, поле status – HTTP статус для відповіді на запит. Для прикладу класу-наслідника BaseRuntimeException розглянемо клас WrongPasswordRuntimeException, який явно вказує свій ExceptionCode (буде використаний при формуванні відповіді на HTTP запит):

```
package edu.diploma.ms.core.exception;
public class WrongPasswordRuntimeException extends BaseRuntimeException {
    public WrongPasswordRuntimeException(String message) {
        super(ExceptionCode.WRONG_PASSWORD_EXCEPTION, message);
    }
}
```

Дана помилка викликається у разі, якщо користувач ввів неправильний пароль для дій з гаманцем. Клас HttpRequestRuntimeExceptionProcessor завдяки анотації ControllerAdvice відповідає за обробку всіх помилок, які стались під час обробки HTTP запиту. Фрагмент коду класу, який відловлює та оброблює всі BaseRuntimeException помилки:

```
@ExceptionHandler(BaseRuntimeException.class)
    public ResponseEntity<ErrorInfo> handleBaseException(BaseRuntimeException e,
HttpServletRequest request) {

        log(request, e);
        ErrorInfo errorInfo = createErrorInfo(e);
        return new ResponseEntity<>(errorInfo,
e.getExceptionCode().getStatus());
    }
```

Анотація ExceptionHandler вказує що саме цей метод варто викликати, коли трапилась така помилка. Метод пише про помилку у консоль методом log, формує об’єкт ErrorInfo методом createErrorInfo та повертає її у відповідь на HTTP запит. Розглянемо реалізацію класів-контролерів на прикладі фрагменту класу UserController, який відповідає за обробку дій користувача з гаманцем:

```
@RestController
@RequestMapping("/api/v1/user")
public class UserController {
    private final UserService userService;
     @PostMapping("/export-wallet")
    public ExportWalletResponse exportWallet(@RequestBody ExportWalletRequest
exportWalletRequest) {
        log.info("Request to export wallet: {}", exportWalletRequest);
        ExportWalletResponse exportWalletResponse =
userService.exportWallet(exportWalletRequest);
        log.info("Request to export wallet processed.");
        return exportWalletResponse;
    }
```

Анотації RestController та RequestMapping вказують фреймворку-Spring, що даний клас має приймати HTTP запити, які мають префікс шляху /api/v1/user. Анотація PostMapping над методом exportWallet вказує, що даний метод обробляє ПОСТ метод з шляхом /export-wallet (після префіксу). Анотація RequestBody перед об'єктом DTO класу ExportWalletRequest вказує, що JSON інформацію з тіла HTTP запиту варто десереалізувати в цей об'єкт. Клас ExportWalletRequest містить в собі ідентифікатор телеграм користувача, та введений ним пароль для перевірки. Фрагмент даного DTO класу:

```
public class ExportWalletRequest {
    private long telegramUserId;
    private String activeWalletPassword;
}
```

Після отримання запиту на експорт гаманця контролер пише про запит у консоль методом log, та передає його у клас-сервіс userService, після чого повертає результат запиту. Далі наведено фрагмент коду з класу UserService, який відповідає за експорт гаманця – метод exportWallet:

```
public ExportWalletResponse exportWallet(ExportWalletRequest
exportWalletRequest) {
        User user = userRepository.
getByTelegramUserId(exportWalletRequest.getTelegramUserId());
if(passwordEncoder.matches(exportWalletRequest.getActiveWalletPassword(),
user.getActiveWalletPassword())) {
            return ExportWalletResponse.builder()
.activeWalletPrivateKey(encryptionService.decrypt(user.getActiveWalletPrivateKey
())).build();
        }
       throw new WrongPasswordRuntimeException("Password does not matches.");
    }
```

Спочатку сервіс за допомогою об’єкта userRepository отримує інформацію про користувача з БД. Після цього, за допомогою об’єкта passwordEncoder, клас якого є частиною Spring Security, перевіряємо відповідність пароля з запиту до хешу пароля з бази даних – якщо вони відрізняються, створюємо помилку WrongPasswordRuntimeException. Якщо пароль вірний – розшифровуємо приватний ключ гаманця користувача (отриманий з БД), та віддаємо його у відповідь користувачу. Розшифруванням займається encryptionService, який секретним ключем з файлу-властивостей за допомогою класу CryptoAES шифрує та розшифровує приватні ключі гаманців користувачів. Наступний фрагмент коду є частиною класу TransactionService, та відповідає за обробку нових транзакцій з блокчейну: співставлення адрес-отримувачів у транзакціях та гаманцях користувачів з БД, формування запиту до be-bot з сповіщеннями про вхідні перекази та оновлення останнього оновленого блоку в базі даних:

```
public void processNewDepositTransactions(List<TransactionTronResponse>
depositTransactions) {
        long lastBlock = depositTransactions.get(0).getBlock();
        Set<String> addressesToCheck =
depositTransactions.stream().map(TransactionTronResponse::getTo).collect(Collect
ors.toSet());
        List<User> usersToNotify =
userRepository.findAllByActiveWalletAddressIn(addressesToCheck);
        Map<String, User> activeAddressToTelegramChatId = usersToNotify.stream()
                .collect(Collectors.toMap(User::getActiveWalletAddress, user ->
user));
        depositTransactions = depositTransactions.stream().filter(trx ->
activeAddressToTelegramChatId.containsKey(trx.getTo()))
                .collect(Collectors.toList());
        List<NewDepositBotNotification> newDepositBotNotifications = new
ArrayList<>();

        depositTransactions.forEach(trx -> {
            User user = activeAddressToTelegramChatId.get(trx.getTo());
            NewDepositBotNotification notification =
NewDepositBotNotification.createOf(user, trx);
            newDepositBotNotifications.add(notification);
        });
        if(!newDepositBotNotifications.isEmpty()) {

botCommunicationService.sendNewDepositNotifications(newDepositBotNotifications);
}
        List<TronBlock> all = tronBlockRepository.findAll();
        if (all.size() == 1) {
            TronBlock tronBlock = all.get(0);
            tronBlock.setLastProcessedBlock(lastBlock);
            tronBlock.setUpdateTimestamp(utcClock.millis());
            tronBlockRepository.save(tronBlock);
```

```
        } else {
tronBlockRepository.save(TronBlock.builder().lastProcessedBlock(lastBlock).build
());}}
```

Фрагмент коду класу BotCommunicationService, який відповідає за спілкування з мікросервісом be-bot (даний тип класів та його реалізація є спільною та ідентичною для всіх мікросервісів):

```
public void sendNewDepositNotifications(List<NewDepositBotNotification>
notifications) {
        URI addressUri = botUriBuilder.getDepositTransactionsNotificationUri();

performRequest(addressUri, HttpMethod.POST, new HttpEntity<>(notifications),
Void.class,
                "Request to bot to send new deposit notifications: {}", new
Object[]{notifications},
                "Request to bot to send new deposit notifications processed.",
"Error when sending new deposit notifications: {}");
    }

    private <T> T performRequest(URI uri, HttpMethod method, HttpEntity<?> body,
Class<T> responseType,
                                 String preLog, Object[] preLogArgs,
                                 String postLog, String errorLog) {
        log.info(preLog, preLogArgs);
        try {
            T responseBody = botRestTemplate.exchange(uri, method, body,
responseType).getBody();
            log.info(postLog);
            return responseBody;
        } catch (ResourceAccessException e) {
            log.error(errorLog, e.getMessage());
            throw new RemoteMsAccessRuntimeException("Error when connecting to
tron: " + e.getMessage());
        }
    }
```

Метод performRequest – типізований метод, який відправляє потрібний запит за вказаною URI, та записує в консоль інформацію про запит. Запит виконується за допомогою об’єкта класу фреймворка Spring – RestTemplate. Метод sendNewDepositTransactions формує потрібну URI за допомогою об’єкта класу BotUriBuilder, та викликає метод performRequest з новими сповіщеннями про вхідні поповнення гаманців користувачів.

Розглянемо основні фрагменти коду мікросервісу be-tron. Фрагмент коду класу TransactionService, який створює транзакцію-переказ коштів, надсилає її в блокчейн та повертає хеш створеної транзакції:

```
public TransactionDto transferCurrency(CurrencyTransferRequest
currencyTransferRequest) {
        try {
            Chain.Transaction transaction =
createAndSignTransferTransaction(currencyTransferRequest);
            String txHash = apiWrapper.broadcastTransaction(transaction);
            return TransactionDto.builder().hash(txHash).build();
        } catch (RuntimeException e) {
            throw new FailedTransactionRuntimeException(e.getMessage());
        }
    }
private Chain.Transaction
createAndSignTransferTransaction(CurrencyTransferRequest
currencyTransferRequest) {
        ByteString rawFromAddress =
ApiWrapper.parseAddress(currencyTransferRequest.getFrom());
        ByteString rawToAddress =
ApiWrapper.parseAddress(currencyTransferRequest.getTo());
        Contract.TransferContract transfer =
Contract.TransferContract.newBuilder()
                .setOwnerAddress(rawFromAddress)
                .setToAddress(rawToAddress)
.setAmount(TronCurrencyConverter.convertToAtomicUnits(currencyTransferRequest.ge
tAmount()))
                .build();
        SECP256K1.KeyPair privateKeyPair =
getKeyPairFromPrivateKey(currencyTransferRequest.getPrivateKey());
        Chain.Transaction transaction =
apiWrapper.blockingStub.createTransaction(transfer);
        TransactionBuilder transactionBuilder = new
TransactionBuilder(transaction);
        if (currencyTransferRequest.getMemo() != null) {
            transactionBuilder.setMemo(currencyTransferRequest.getMemo());
        }
        transaction = transactionBuilder.build();
        transaction = apiWrapper.signTransaction(transaction, privateKeyPair);
        return transaction;
    }
```

Створена транзакція створюється на основі об’єкта DTO класу HTTP запиту CurrencyTransferRequest, та підписується і надсилається в блокчейн за допомогою об’єкта класу бібліотеки trident – ApiWrapper. Фрагмент коду, який відповідає за пошук нових вхідних переказів користувачам бота:

```
private void search() throws InterruptedException {
        long lastProcessedBlock =
coreCommunicationService.getBlock().getLastProcessedBlock();

        BlockDataTronScanResponse latestBlockNumber =
tronScanCommunicationService.getLatestBlockNumber();

        long lastBlockchainBlockToProcess = latestBlockNumber.getNumber() - 19;
TransactionDto emptyTransactionForBlockUpdate = TransactionDto.builder()
                .to("")
                .block(latestBlockNumber.getNumber() - 19)
                .build();
```

```
        if(lastProcessedBlock == 0) {
coreCommunicationService.sendNewDepositTransactions(Collections.singletonList(em
ptyTransactionForBlockUpdate));
            return;
        }
        List<TransactionHistoryTronScanResponse.TransactionData>
transactionDataList = new ArrayList<>();
        while (lastBlockchainBlockToProcess > lastProcessedBlock) {
            TransactionHistoryTronScanResponse transactionsByBlock =
                    tronScanCommunicationService.getTransactionsByBlock(50, 0,
lastBlockchainBlockToProcess);
            int collectedTransactions = 50;
            fillListWithTrxTransactions(transactionDataList,
transactionsByBlock);
            while (collectedTransactions < transactionsByBlock.getTotal()) {
                transactionsByBlock = tronScanCommunicationService
                        .getTransactionsByBlock(50, collectedTransactions,
lastBlockchainBlockToProcess);
                fillListWithTrxTransactions(transactionDataList,
transactionsByBlock);
                collectedTransactions = collectedTransactions + 50;
            }
            lastBlockchainBlockToProcess--;
            TimeUnit.MILLISECONDS.sleep(500);
        }
        List<TransactionDto> depositTransactions = transactionDataList.stream()
                .map(TransactionConverter::convertTronTransaction)
                .collect(Collectors.toList());
        if(depositTransactions.isEmpty()) {
            depositTransactions.add(emptyTransactionForBlockUpdate);
        }
        log.info("New trx transactions amount: {}", transactionDataList.size());
        System.out.println(depositTransactions.get(0).getBlock());
  coreCommunicationService.sendNewDepositTransactions(depositTransactions);}
```

Даний фрагмент коду викликається раз в 5 секунд методом searchNewTrxTransferTransactions, який позначений спеціальною анотацією Scheduled, яка й вказує необхідність таких викликів фреймворку Spring. Метод search спочатку звертається до be-core за останнім обробленим блоком. Якщо блок має значення 0, це означає що програма запущена вперше – метод надсилає до be-core під виглядом вхідної транзакції останній підтверджений блок з блокчейну, який буде записаний в be-core як останній оброблений. В іншому випадку, метод дістає з блокчейну всі нові підтверджені блоки, фільтрує серед них транзакції-перекази TRX та відправляє їх до be-core. Фрагмент коду, який генерує нові акаунти для користувачів:

```
public TronAccountDto generateNewAccount() {

        List<String> list = ApiWrapper.generateAddress();
```

```
        String privateKey = list.get(1);
        String publicKeyBase58 =
Base58Check.bytesToBase58(ApiWrapper.parseHex(list.get(0)).toByteArray());
        return new TronAccountDto(privateKey, publicKeyBase58);}
```

Розглянемо фрагменти основних модулів be-bot. Фрагмент класу MessageConfigurationStrorage, а саме функція Supplier, яка повертає налаштування інтерфейсу для повідомлення з інформацією про акаунт у блокчейні TRON:

```
public static final Supplier<MessageConfiguration> ACCOUNT = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().textGenerator(SELECTED_ACCOUNT_TEXT).bu
ild()).callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\u2b05\ufe0f Назад").build())
.id("BACK_TO_SEARCH").next(Keyboard.SEARCH_ACCOUNT)
.inputValidator(ADDRESS_VALIDATOR).action(SEARCH_ACCOUNT_ACTION)
.multiClickable(true).build())))).build();
```

Кожна конфігурація містить частину для налаштування тексту (MessageTextConfiguration), поле keyboardType (визначає чи це інтерфейс головного чи вкладеного меню) та вкладені списки ButtonConfiguration – конфігурацій кнопок. Налаштування тексту може містити готовий статичний текст (поле text), заготовку для тексту та функцію з генерацією аргументів для цього шаблону (поля message та argGenerator) або метод для повноцінної динамічної генерації тексту (поле textGenerator). ButtonConfiguration містить аналогічне поле-клас ButtonLabelConfiguration для налаштування тексту на кнопці, а також наступні поля: String id – ідентифікатор кнопки, який ми отримаємо від API Telegram, якщо користувач натисне цю кнопку; enum TypeOfButton type – тип кнопки, дорівнює значенню PAGINATION якщо дана кнопка є місцем для динамічного заповнення іншими кнопками (наприклад, історією транзакцій), яке використовується та заповнюється на етапі формування клавіатури; enum Action action – поле містить функцію, яку варто викликати при натиску на кнопку; enum InputValidator inputValidator – функція, яка перевіряє введені дані на коректність (наприклад, сума переказу більша за 0); enum ButtonDynamicValidator dynamicValidator – функція, яка вказує чи має дана

кнопка бути показана користувачу; enum Keyboard поля next та nextAfterInput – визначають наступну клавіатуру після натиску кнопки, та клавіатуру яка буде наступною після коректного введеної інформації; String relatedObjectField – текстовий шлях до певного поля в сесії користувача, необхідні при взаємодії з кнопкою; boolean multiClickabe – визначає, чи можна натиснути кнопку декілька разів підряд. Вище вказані функції, такі як Action, приймають параметром клас ResponseMessageWrapper – клас-обгортку для сесії користувача та інших тимчасових полів, які потрібні для формування повідомлення-відповіді. Клас UserSession є сесією користувача – він зберігає інформацію про те, на якій клавіатурі знаходиться користувач, вхідні дані від користувача та поля-запити до інших мікросервісів, які користувач заповнює:

```
public class UserSession {

    private TelegramUserInfo telegramUserInfo;
    private String callbackData;
    private String inputData;
    private Integer lastInlineKeyboardMessageId;
    private Integer lastReplyKeyboardMessageId;
    private Integer lastMessageId;

    private PaginationState paginationState;
    private InputType inputType;

    private MessageConfiguration messageConfiguration;
    private ButtonConfiguration activeButton;
    private SessionState state;

    private String activeWalletAddress;

    private AccountResourcesTronResponse selectedAccount;
    private TransactionTronResponse selectedTransaction;

    private NewWalletCoreRequest createWalletRequest;
    private NewWalletCoreRequest importWalletRequest;

    private CurrencyTransferCoreRequest currencyTransferCoreRequest;
    private FreezeBalanceCoreRequest freezeBalanceCoreRequest;

}
```

Копією даного класу є UserSessionModel, який зберігається в БД бота задля того, щоб після перезапуску бота всі сесії були збережені. Приклад функції Action, яка відповідає за пошук акаунту:

```
public static final Consumer<ResponseMessageWrapper> SEARCH_ACCOUNT_ACTION = m -
> {TronCommunicationService communicationService =

m.getCtx().getBean(TronCommunicationService.class);
UserSession userSession = m.getUserSession();
        AccountResourcesTronResponse accountResources =
communicationService.getAccountResources(userSession.getInputData());
        userSession.setSelectedAccount(accountResources);
        MessageConfigurationStorage.setKeyboardToSession(userSession,
Keyboard.ACCOUNT);

        userSession.getActiveButton().deactivate();
    };
```

Приклад функції InputValidator, яка перевіряє чи є введений текст сумою:

```
public static final Consumer<ResponseMessageWrapper> AMOUNT_VALIDATOR = m ->
{validateTextInputType(m);
        try {double amount =
Double.parseDouble(m.getUserSession().getInputData());

            if (amount <= 0 || Double.compare(0.000001, amount) > 0) {
                throw new ValidationErrorException(String.format("Validation
failed -> amount didn't change to %s", m.getUserSession().getInputData()),
"Введений текст не сумою - потрібно ввести додатнє число, мінімальне значення -
0.000001."); }} catch (NumberFormatException e) {
throw new ValidationErrorException(String.format("Validation failed -> amount
didn't change to %s", m.getUserSession().getInputData()), "Введений текст не
сумою - потрібно ввести додатнє число, мінімальне значення - 0.000001.");}};
```

Приклад функції TextGenerator, яка відповідає за повідомлення з курсом:

```
public static final Function<ResponseMessageWrapper, String> RATE_TEXT = m ->
getTronRateMessage(m.getCtx());
    private static String getTronRateMessage(ApplicationContext ctx) {

CurrencyRatesStorage currencyRatesStorage =
ctx.getBean(CurrencyRatesStorage.class);

        return formatMessage("Курс 1 TRX до гривні: {0}\nКурс 1 TRX до долара:
{1}\n", toPlainString(currencyRatesStorage.getUahRate()),
                toPlainString(currencyRatesStorage.getUsdRate()));
    }
```

Фрагмент коду класу BotGateway, який отримує нові повідомлення від користувачів (параметр update), блокує доступ до одночасної обробки дії за допомогою об’єкта lockRegistry та ідентифікатора користувача, отримує сесію користувача з SessionService та передає її з повідомленням до об’єкту класу який оброблює дії користувача – UserUpdateProcessor:

```
public Reply replyToMessage() {Consumer<Update> consumer = u ->
updateProcessorExecutor.execute(() -> {
            Long chatId = u.getMessage().getChatId();
            lockRegistry.obtain(chatId).lock();
            try {
                UserSession userSession = sessionService.prepareSession(u,
chatId, u.getMessage().getText());
                userUpdateProcessor.processInputMessage(u, userSession);
                sessionService.update(userSession, chatId);
            } finally {lockRegistry.obtain(chatId).unlock();}
        });return Reply.of(consumer, MESSAGE);}
```

Фрагмент класу MessageSender, який відправляє повідомлення з клавіатурами головного або вкладеного меню користувачам:

```
public class MessageSender {

    private final BotGateway botGateway;

    public void sendMessage(ResponseMessageWrapper m) {

        try { if (m.getUserSession().getMessageConfiguration().getKeyboardType()
== KeyboardType.INLINE) {sendInlineMessage(m);} else {sendReplyMessage(m); }
        } catch (TelegramApiRequestException e) {
            String message = String.format("Error sending message to user %d,
code %d: %s: %s",
                    m.getUserId(), e.getErrorCode(), e.getMessage(),
e.getApiResponse());
            log.error(message);throw new TelegramApiRequestExceptionWrapper(
e, message); } catch (TelegramApiException e) {
            String message = String.format("Error sending message to user %d:
%s", m.getUserId(), e.getMessage());
            log.error(message); throw new RuntimeException(message, e);
        }}

    private void sendInlineMessage(ResponseMessageWrapper m) throws
TelegramApiException {

if(m.getMessageProcessStrategy() == MessageProcessStrategy.CREATE_NEW) {Integer
messageId =
botGateway.execute(MessageSenderConverter.prepareSendInlineMessage(m)).getMessag
eId();
            UserSession userSession = m.getUserSession();

             userSession.setLastMessageId(messageId);

if (!m.getCallbacks().isEmpty()) {
                userSession.setLastInlineKeyboardMessageId(messageId); }} else
{botGateway.execute(MessageSenderConverter.prepareEditMessageText(m));}}
    private void sendReplyMessage(ResponseMessageWrapper m) throws
TelegramApiException {
        Integer messageId =
botGateway.execute(MessageSenderConverter.prepareSendReplyMessage(m)).getMessage
Id(); m.getUserSession().setLastReplyKeyboardMessageId(messageId);
}
```

Код (лістинг) програми подано у додатку В.

3.3 Керівництво користувача

Коли користувач вперше відкриє діалог з ботом, він ініціює розмову натиснувши кнопку “start”, після чого від його імені боту буде надіслана команда “/start”. У відповідь бот покаже користувачу повідомлення з головним меню без дій з активним гаманцем, оскільки він відсутній (рисунок 3.1).

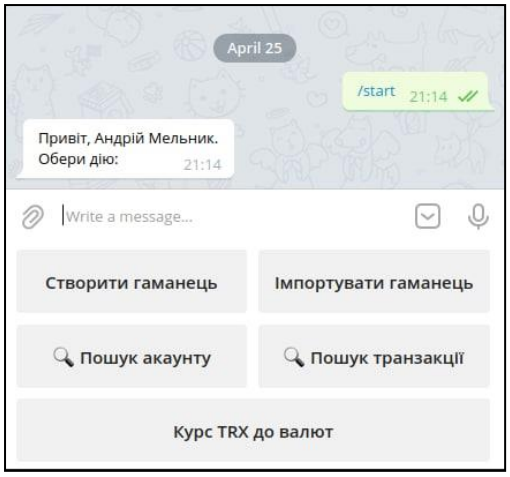


Рисунок 3.1 – Головне меню бота

Для створення нового гаманця, користувач повинен натиснути відповідну кнопку, після чого побачить меню для створення гаманця (рисунок 3.2).

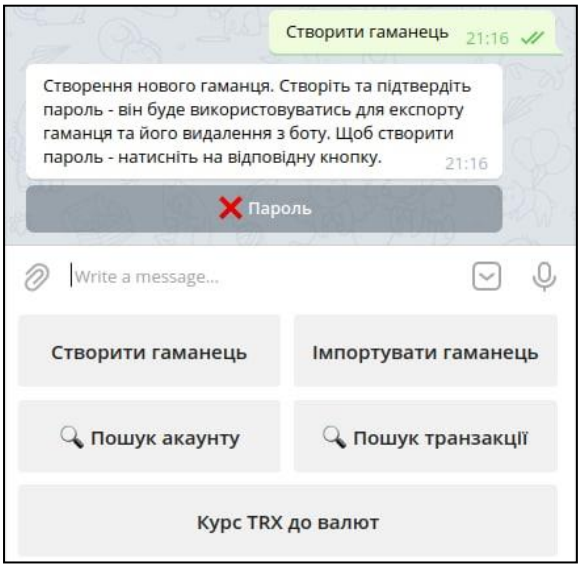


Рисунок 3.2 – Створення гаманця

Для створення паролю, користувач повинен натиснути на відповідну кнопку, та ввести пароль. Після цього користувач повинен підтвердити пароль, натиснувши відповідну кнопку “Підтвердити пароль” та повторно ввівши пароль. Після заповнення цих полів, користувачу буде доступна кнопка для створення гаманця (рисунок 3.3). Натиснувши на кнопку, буде створений гаманець та показане головне меню з повним функціоналом (рисунок 3.4). Для імпорту гаманця, необхідно на головному меню натиснути відповідну кнопку, після чого аналогічним чином заповнити пароль. Натиснувши на кнопку “Імпортувати гаманець”, користувач має ввести приватний ключ від свого гаманця, після чого йому буде показане те саме меню з повним функціоналом.

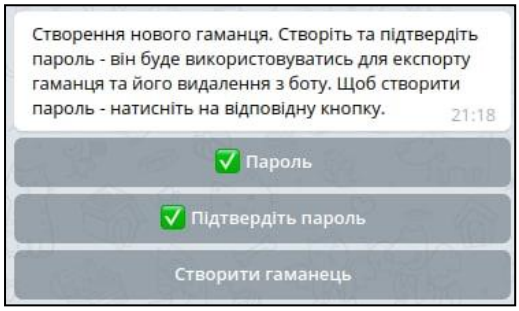


Рисунок 3.3 – Створення гаманця (заповнені дані з кнопкою для створення)

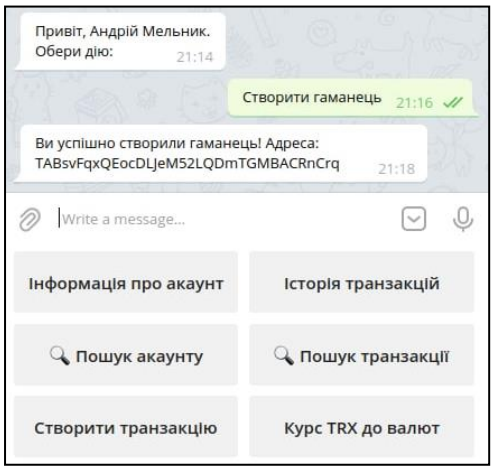


Рисунок 3.4 – Успішне створення гаманця з повним головним меню

Щоб переглянути інформацію про певну транзакцію, треба натиснути кнопку “Пошук транзакції”, після чого ввести хеш транзакції. Користувачу буде

показана інформація про транзакцію з можливістю повернутись назад до пошуку (рисунок 3.5). Аналогічним чином відбувається пошук акаунту – інформація про знайдений акаунт зображений на рисунку 3.6.

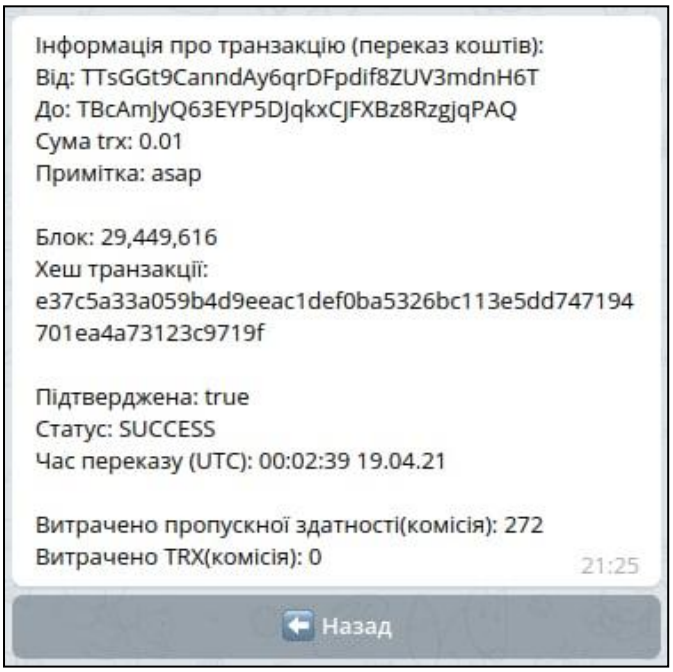


Рисунок 3.5 – Інформація про транзакцію з пошуку

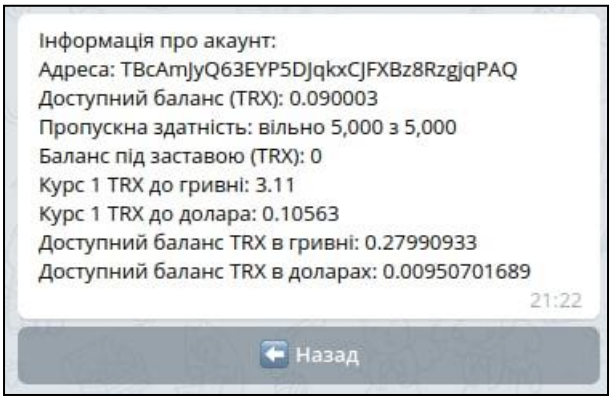


Рисунок 3.6 – Інформація про акаунт з пошуку

Меню історії транзакцій (можна перейти з головного меню) зображене на рисунку 3.7. Меню містить скорочену інформацію у текстовому форматі про кожну транзакцію, теперішню та загальну кількість сторінок, та кнопку для кожної транзакції на сторінці, після натиску якої, користувач побачить інформацію про транзакцію (аналогічно до рисунка 3.5).

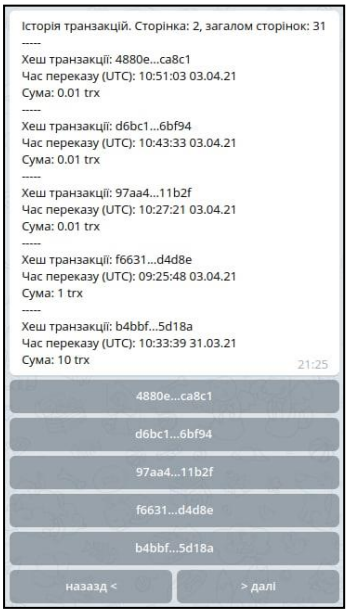


Рисунок 3.7 – Історія транзакцій

Кнопка “Інформація про акаунт” відкриває меню власного гаманця. Якщо гаманець новий та не активований, користувач побачить в меню текст про необхідність активації (рисунок 3.8). Якщо акаунт активований, текст буде відповідним до рисунку 3.6. Для експорту чи видалення гаманця з бота, необхідно натиснути відповідну кнопку в меню власного гаманця та ввести пароль. Після видалення користувач побачить відповідне повідомлення, після експорту – повідомлення з приватним ключем.

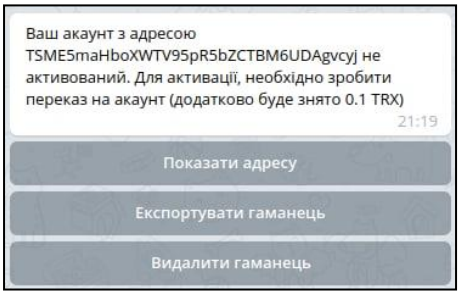


Рисунок 3.7 – Інформація про власний акаунт

Для створення транзакції, необхідно натиснути відповідну кнопку, після чого обрати тип транзакції (переказ, застава тощо). Після обраного типу, необхідно заповнити відповідні поля (рисунок 2.11) та натиснути кнопку

“підтвердити”. Після цього, користувачу буде показана ціна транзакції в балах пропускної здатності з кнопкою-підтвердженням створення транзакції (рисунок 3.8). Після підтвердження, користувачу буде повернений хеш (ідентифікатор) створеної транзакції. На рисунку 3.9 зображено сповіщення користувачу про новий вхідний переказ.

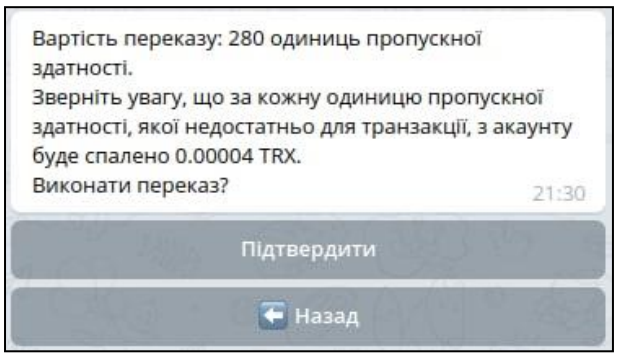


Рисунок 3.8 – Підтвердження створення транзакції

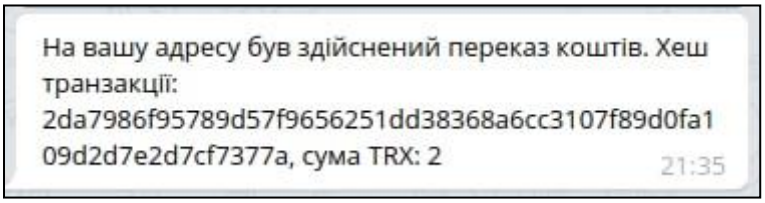


Рисунок 3.9 – Сповіщення про вхідний переказ

3.4 Вимоги до технічних та програмних засобів

Для користувача бота необхідний клієнт Telegram – у вигляді сторінки в браузері, комп’ютерного чи мобільного додатку тощо. Серверне забезпечення, де буде запущена система повинне задовольняти наступні мінімальні вимоги:

- 2-ядерний 64-розрядний процесор з частотою 2 ГГц;
- 4 Гб оперативної пам’яті;
- 40 Гб дискового простору;
- операційна система Ubuntu 18;
- підключення до інтернету зі швидкістю 100 Мб/с.

3.5 Розгортання та встановлення системи

Для запуску та роботи системи, необхідно встановити наступні додатки: JRE (Java Runtime Environment) версії 11 або новішої для запуску компільованих мікросервісів (jar файли), MongoDB (БД) та GNU Screen – додаток для Linux систем, який дозволяє запускати програми з терміналу в різних процесах. Для встановлення необхідно виконати наступні команди в консолі (кожна стрічка – окрема команда):

```
sudo apt update && apt install -y openjdk-11-jre-headless
sudo apt-get install -y screen && sudo apt install -y mongodb
```

Щоб створити бота, його необхідно зареєструвати в Telegram за допомогою офіційного бота BotFather. На рисунку 3.10 зображений процес створення нового бота – реєстрація його ім’я та отримання токена-ключа.

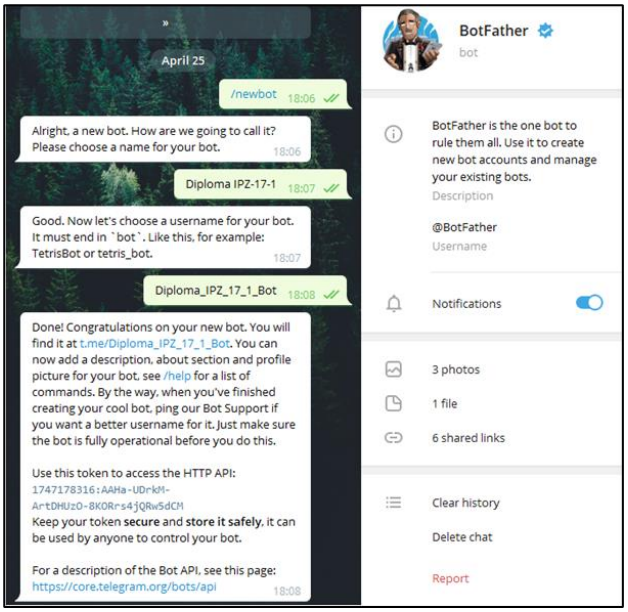


Рисунок 3.10 – Реєстрація бота

Після встановлення вказаних компонентів та реєстрації бота, потрібно створити базу даних та її користувача. Для цього в терміналі потрібно послідовно виконати наступні команди:

```
mongo
use diploma;
db.createUser({user:” user”, pwd:” pass”, roles:[{role:”readWrite”, db:”
diploma”}]});
use diploma_bot;
db.createUser({user:”bot_user”, pwd:” bot_pass”, roles:[{role:”readWrite”, db:”
diploma_bot”}]});
exit
```

Далі необхідно запустити систему – необхідно запустити кожен jar файл мікросервісу в окремому процесі. Приклад запуску сервісу be-core з терміналу – створимо окремий Screen-процес, та запустимо в ньому jar файл мікросервісу:

```
screen -S core && java -jar be-core.jar
```

Після цього, потрібно виконати комбінацію Ctrl-a d щоб від’єднатись від процесу, залишаючи його в стані виконання. Для be-tron виконуємо аналогічні дії (змінюючи назви процесів в першій команді та jar-файли в другій). Для be-bot необхідно передати боту токен бота та його назву, отримані від BotFather, а тому команда для його запуску виглядає наступним чином:

```
java -jar be-bot.jar --bot.username=Diploma_IPZ_17_1_Bot --bot.token=
1747178316:AAHa-UDrkM-ArtDHUzO-8KORrs4jQRw5dCM --bot.creator.id=1747178316
```

Для зупинки роботи системи, необхідно виконати наступну команду для всіх мікросервісів (замість core ввести назви інших модулів):

```
screen -S core -X quit
```

В даному розділі було проведене проектування окремих мікросервісів, визначені основні модулі кожного з них та побудована їх взаємодія. Було виконано розробку програмних модулів, описані їх особливості за допомогою фрагментів коду. Було створено інструкцію користувача, визначені вимоги до технічних та програмних засобів для запуску системи, а також описаний процес розгортання та встановлення системи.

4. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Вибір та обґрунтування методів тестування додатку

При виборі методів тестування варто враховувати те, що розроблена система складається з кількох окремих програм мікросервісів. Загальноприйнятою практикою в такому випадку є тестування кожного мікросервісу окремо від інших за допомогою модульних та інтеграційних тестів. Модульне (unit) тестування – перший етап тестування системи, при якому відбувається перевірка кожного модуля програми окремо. В даному контексті модулем є найменша частина програми, яку можна протестувати – клас, метод класу, комбінація декількох методів одного класу тощо. Дане тестування зазвичай виконується програмістом який написав дані класи, але також може бути виконаним тестувальником. У цьому випадку, код тестується за принципом “білої скриньки”, тобто відповідальна особа знає внутрішню структуру модуля та його очікувану поведінку. Інтеграційне (integration) тестування – наступний етап, який відбувається після модульного тестування. На цій стадії протестовані раніше модулі розглядаються як взаємодіючі між собою частини одного цілого. Зазвичай в контексті мікросервісів дана фаза тестування перевіряє систему за принципом “чорного ящика”, тобто очікується певна вихідна інформація (відповідь на запит, запис у БД тощо) у відповідь на вхідні дані (HTTP запит, нові транзакції в блокчейні тощо). Важливим є те, що на даних етапах тестування зовнішні залежності, такі як БД чи інші мікросервіси, заміняються об’єктами-макетами (mock), оскільки правильна поведінка програми від них не залежить, і відповідно такі залежності не повинні бути фактором успішності тесту.

Такий підхід дозволяє полегшити процес тестування, одразу обмежуючи відповідальність тестів функціональним призначенням саме цього мікросервісу, а не комбінацією декількох модулів системи. Дана декомпозиція дозволяє швидше знайти та локалізувати помилки в коді, які необхідно виправити. Але, такий підхід не надає можливості перевірити роботу сукупності модулів ПЗ як

єдиного цілого (включно з користувацьким інтерфейсом). Для цього виконаємо системне тестування, при якому вся система повинна бути інтегрованою. Оперування системою буде відбуватись на рівні інтерфейсу користувача, тест-кейси повинні бути побудовані на основі вимог до ПЗ та інструкції користувача. Зазвичай даний підхід до перевірки, як і інтеграційне тестування, слідує принципу “чорного ящику” – внутрішні процеси роботи системи не враховуються при тестуванні. Також використаємо один зі способів тестування продуктивності системи – навантажувальне (load) тестування, яке дозволяє оцінити здатність роботи ПЗ при певному навантаженні (наприклад, одночасне використання декількома користувачами), час обробки запитів від користувача тощо. Такий підхід може використовуватись як при інтегрованій системі, так і для її окремих модулів.

Для проведення модульного та інтеграційного тестування на мові Java використаємо наступні інструменти: JUnit, Mockito, AssertJ, Flapdoodle Embedded MongoDB та Spring Boot Starter Test – модуль Spring, який додає необхідний функціонал для написання модульних та інтеграційних тестів для Spring Web. Також, даний модуль вже містить в собі три перші інструменти зі списку. JUnit – фреймворк для автоматизованого тестування Java програм. Станом на 2013 рік серед 10000 проектів на GitHub, даний інструмент використовувався у 30.7% з них. Основними анотаціями є Test (позначає, що даний метод є окремим тестом) та BeforeAll (вказує метод, який має викликатись перед початком роботи тестів), які вказують фреймворку що саме та з якою метою варто викликати. Mockito – фреймворк для створення mock об’єктів, які використовуються для імітації роботи тих чи інших класів. Зазвичай використовується для моделювання комунікації з іншими програмами, звертань до БД тощо. AssertJ – бібліотека, яка надає зручний інтерфейс для порівняння очікуваних результатів тестування з реальними. Flapdoodle Embedded MongoDB – бібліотека яка надає можливість створити та запустити вбудовану в програму MongoDB під час тестів, тому її можна використовувати для перевірки роботи ПЗ з БД. Таким чином, даний засіб

забирає БД зі списку зовнішніх залежностей. Для навантажувального тестування було обрано використовувати Apache JMeter – спеціалізоване ПЗ, розроблене для створення та виконання тестів такого призначення.

4.2 Розробка тестових сценаріїв

Розпочнемо з модульного тестування. Перевіримо роботу окремих класів мікросервісів на коректність обробки даних. Деякі тестові сценарії модульних тестів наведені в таблиці 4.1.

Таблиця 4.1 – Тестові сценарії модульних тестів

Іденти-фікатор	Мікро-сервіс	Модуль	Вихідні дані	Очікуваний результат
1	2	3	4	5
T-M-1	be-tron	Конвертування SUN в Tronix	Сума коштів у SUN	Сума коштів конвертована у Tronix
T-M-2	be-tron	Створення акаунту	Пароль для нового гаманця	Приватний ключ та адреса нового акаунту
T-M-3	be-tron	Переказ коштів	Приватний ключ відправника, адрес відправника, сума, адрес отримувача, примітка	Виклик бібліотеки trident для переказу у блокчейні, повернення хешу нової транзакції. Якщо балансу або ресурсів акаунту недостатньо, виникла проблема з TRON – повернути помилку.
C-M-1	be-core	Видача останнього обробленого блоку	відсутні	Номер останнього обробленого блоку транзакцій, збережений в БД
B-M-1	be-bot	Підготовка тексту повідомлення	Конфігурація тексту, сесія користувача з параметрами для тексту	Сформований текст

Кінець таблиці 4.1

1	2	3	4	5
B-M-2	be-bot	Підготовка клавіатури повідомлення	Конфігурація клавіатури, сесія користувача	Відбір потрібних кнопок. Формування тексту кнопок. Впорядкування кнопок.
B-M-3	be-bot	Обробка вхідного повідомлення користувача	Вхідне текстове повідомлення, сесія користувача	Визначення дії користувача. Перевірка введеної інформації на правильність. Обробка дії з введеним текстом.

Фрагмент коду нижче є фрагментом тесту T-M-1, який перевіряє коректність роботи класу, який відповідає за конвертування SUN в Tronix:

```
@Test void shouldProperlyConvertAtomicUnits(){Assertions.assertThat
(TronCurrencyConverter.convertAtomicUnits(1_000_000_0)).isEqualTo(10);}
```

Перейдемо до інтеграційного тестування. Протестуємо функції програми які використовують найбільше різних компонентів, відповідно мають більшу ймовірність помилок в порівнянні з окремими модулями. наприклад пошук нових вхідних переказів, переказ коштів тощо. Певні сценарії інтеграційних тестів наведені в таблиці 4.2.

Таблиця 4.2 – Тестові сценарії інтеграційних тестів

Іденти-фікатор	Мікро-сервіс	Модуль	Вихідні дані	Очікуваний результат
1	2	3	4	5
T-I-1	be-tron	Пошук нових вхідних транзакцій	відсутні	Звернення до be-core за номером останнього обробленого блоку. Пошук нових транзакцій, фільтрування транзакцій переказів, та їх передача до be-core.

Кінець таблиці 4.2

1	2	3	4	5
C-I-1	be-core	Створення акаунту	Ідентифікатор користувача Telegram, новий пароль для гаманця	Звернення до be-tron для отримання нового гаманця. Збереження даних користувача та гаманця до БД. Повернення адреси гаманця.
C-I-2	be-core	Обробка нових вхідних переказів	Транзакції-перекази	Пошук в БД користувачів-отримувачів. Співставлення з транзакціями. Відправка сповіщень до be-bot. Збереження останнього обробленого блоку.

Фрагмент коду нижче є частиною тесту C-I-2:

```
@Test
    void shouldCreateWalletWhenRequestComesThenSuccessfullyCreate() throws
Exception {
BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder(10);
TronAccountTronResponse tronResponse = new TronAccountTronResponse("privateKey",
"publicKey");

        mockRestTemplateSuccessResponse(tronServiceServer, HttpMethod.GET,
tronUriBuilder.getGenerateAccountUri(), tronResponse);

        NewWalletRequest request =
NewWalletRequest.builder().chatId(123).telegramUserId(321)
                .activeWalletPassword("password").build();

        sendPostRequest("/api/v1/user/create-new-wallet",
request).andExpect(status().isOk())
.andExpect(jsonPath("address").value(tronResponse.getAddress()));

Assertions.assertThat(userRepository.findAll()).hasSize(1);
User createdUser = userRepository.findAll().get(0);

Assertions.assertThat(passwordEncoder.matches(request.getActiveWalletPassword(),
createdUser.getActiveWalletPassword())).isTrue();
Assertions.assertThat(encryptionService.decrypt(createdUser.getActiveWalletPriva
teKey())).isEqualTo(tronResponse.getPrivateKey());
Assertions.assertThat(createdUser.getTelegramUserId()).isEqualTo(request.getTele
gramUserId());
Assertions.assertThat(createdUser.getChatId()).isEqualTo(request.getChatId());}
```

Системне тестування буде виконуватись з використанням інтерфейсу користувача та інструкції, написаній в попередньому розділі. На даному етапі

перевіряється взаємодія всіх мікросервісів, їх робота зі сторонніми сервісами (TronScan, Binance, Telegram API тощо). Частина тестових сценаріїв системного тестування наведені в таблиці 4.3.

Таблиця 4.3 – Тестові сценарії системних тестів

Іденти-фікатор	Модуль	Вихідні дані	Очікуваний результат
ST-1	Створення нового гаманця	1. Користувач ініціював розмову з ботом. 2. Користувач натиснув кнопку “Створити гаманець”. 3. Користувач натиснув кнопку “Пароль” та ввів нове значення. 4. Користувач натиснув кнопку “Підтвердити пароль” та повторно ввів пароль. 5. Користувач натиснув кнопку “Створити гаманець”.	1. Користувачу у відповідь надсилається головне меню. 2. Користувачу надсилається клавіатура для створення гаманця. 3. Система додала до інтерфейсу кнопку для підтвердження паролю. 4. Система додала до інтерфейсу кнопку для створення гаманця. 5. Користувачу показана адреса його нового гаманця, змінене головне меню для взаємодії з гаманцем.
ST-2	Переказ коштів	1. Користувач натиснув кнопку “Створити транзакцію”. 2. Користувач натиснув кнопку “Переказ коштів” 3. Користувач натиснув кнопку “Сума TRX” та ввів нове значення. 4. Користувач натиснув кнопку “Адрес отримувача” та ввів нове значення. 5. Користувач натиснув кнопку “Підтвердити” 6. Користувач натиснув наступну кнопку “Підтвердити”	1. Користувачу у відповідь надсилається клавіатура з вибором типу транзакції. 2. Користувачу надсилається інтерфейс для створення переказу. 3. Система оновила повідомлення з вхідними даними. 4. Система оновила повідомлення з вхідними даними. 5. Користувачу показана вартість переказу коштів з кнопкою для повторного підтвердження. 6. Користувач отримав хеш створеної транзакції.

Тестування навантаженням було використано для перевірки одночасного створення 10 переказів від одного гаманця іншим – найбільш вимоглива задача в

системних ресурсах. Така кількість переказів була обрана як кількість, яку важко досягнути при звичайному користуванні системою, отже якщо система витримає дане навантаження – при звичайному користуванні система буде функціонувати без помилок. При даному тестуванні були задіяні мікросервіси be-core та be-tron, оскільки вони виконують підготовку та відправку транзакцій до блокчейну. На рисунках 4.1 та 4.2 зображенні налаштування тестового сценарію – кількості HTTP запитів та конфігурація самого HTTP запиту.

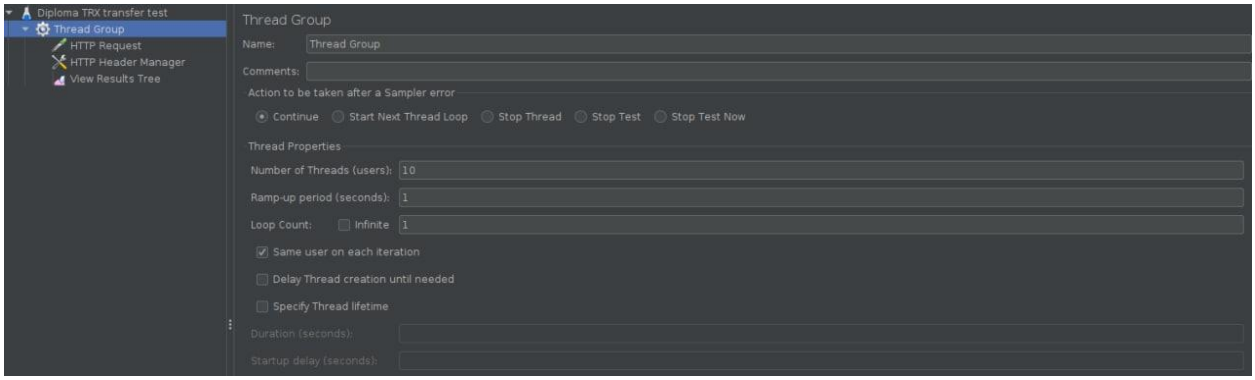


Рисунок 4.1 – Налаштування кількості HTTP запитів для тестового сценарію

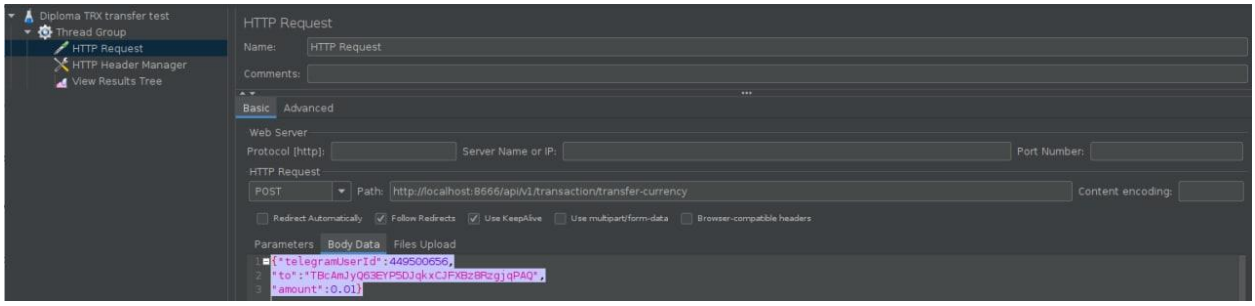


Рисунок 4.2 – Налаштування HTTP запиту для тестового сценарію

4.3 Аналіз результатів тестування системи

Дані тестові сценарії виконувались у порядку охоплення функціоналу системи – від найменших (модульних) до найбільших (системних) тестів, оскільки таким чином легше знайти та виправити помилки, переконуючись у справному функціонуванні менших модулів. На рисунку 4.3 зображений

фрагмент інтерфейсу IDE, який показує результат виконання модульних тестів (в даному випадку, тестового сценарію Т-М-1).

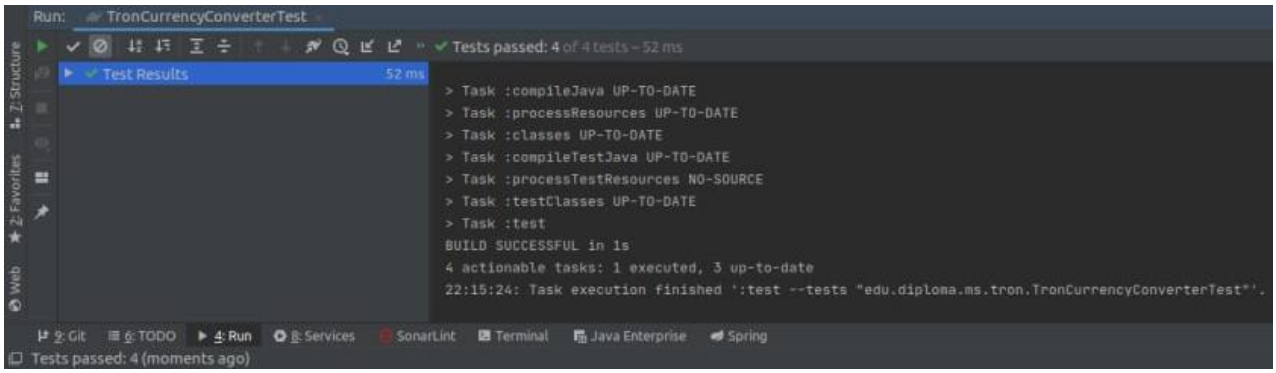


Рисунок 4.3 – Результат тестового сценарію Т-М-1

Інтеграційні тестові сценарії виконувались аналогічним чином до модульних тестів, за допомогою IDE. На рисунку 4.4 зображене виконання та результат навантажувального тесту для одночасного створення переказів (кожен з 10 запитів успішно створив переказ та отримав у відповідь на запит ідентифікатор транзакції та іншу інформацію про переказ):

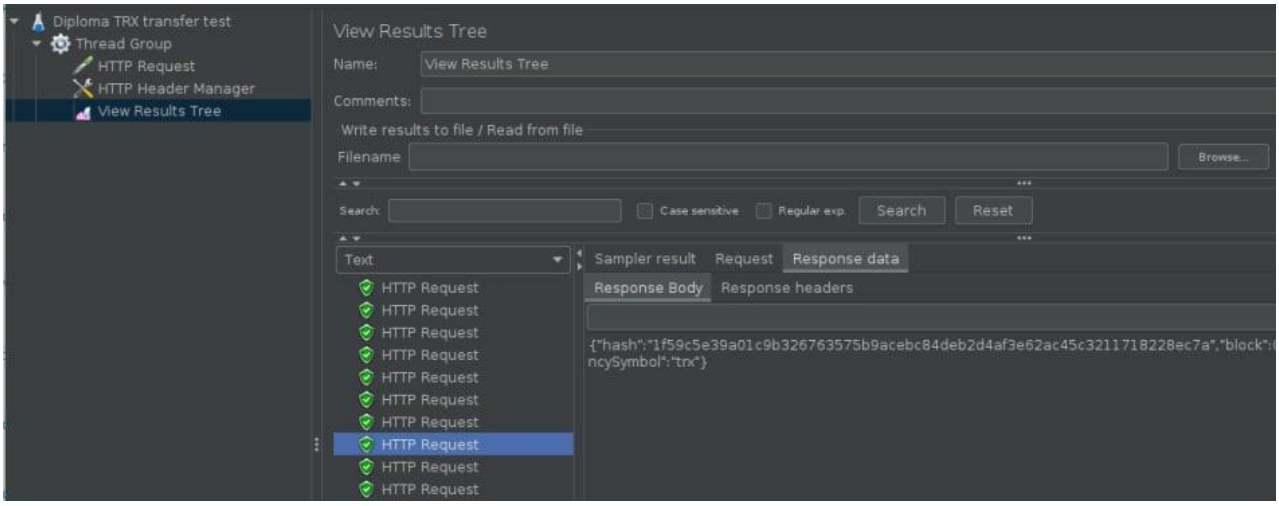


Рисунок 4.4 – Результат навантажувального тесту

Результати тестування за допомогою створених сценаріїв, певні вхідні дані для сценаріїв наведені в таблиці 4.4.

Таблиця 4.4 – Результати тестових сценаріїв

Іденти-фікатор	Опис	Вхідні значення	Вихідні значення	Результат
1	2	3	4	5
Т-М-1	Конвертування SUN в Tronix	1039005	1.039005	Правильно
Т-М-2	Створення акаунту	Відсутні	Приватний ключ та адреса нового акаунту	Правильно
Т-М-3	Переказ коштів	Приватний ключ відправника, адрес відправника, сума, адрес отримувача, примітка	Виклик бібліотеки trident для переказу у блокчейні, повернення хешу нової транзакції. Якщо балансу або ресурсів недостатньо – повернути помилку.	Правильно
С-М-1	Видача останнього обробленого блоку	відсутні	Останній оброблений блок, збережений в БД	Правильно
В-М-1	Підготовка тексту повідомлення	Конфігурація тексту, сесія користувача	Сформований текст	Правильно
В-М-2	Підготовка клавіатури повідомлення	Конфігурація клавіатури, сесія користувача	Відбір потрібних кнопок. Формування тексту кнопок. Впорядкування кнопок.	Правильно
В-М-3	Обробка вхідного повідомлення користувача	Вхідне текстове повідомлення, сесія користувача	Визначення дії користувача. Перевірка введеної інформації на правильність. Обробка дії з введеним текстом.	Правильно

Кінець таблиці 4.4

1	2	3	4	5
T-I-1	Пошук нових вхідних транзакцій	відсутні	Звернення до be-core за останнім обробленим блоком. Пошук нових транзакцій та їх передача до be-core.	Правильно
C-I-1	Створення акаунту	Ідентифікатор користувача Telegram, новий пароль для гаманця	Звернення до be-tron для отримання нового гаманця. Збереження даних користувача та гаманця до БД. Повернення адреси гаманця.	Правильно
C-I-2	Обробка нових вхідних переказів	Транзакції-перекази	Пошук в БД користувачів-отримувачів. Співставлення з транзакціями. Відправка сповіщень до be-bot. Збереження останнього обробленого блоку.	Правильно
ST-1	Створення нового гаманця	Пароль “Qwery1!!”, підтвердження паролю	Користувачу показана адреса його нового гаманця	Правильно
ST-2	Переказ коштів	Адрес отримувача “TBcAmJyQ63EYP5 DJqkxCJFXBz8RzgjqPAQ”, сума TRX – 0.01	Хеш створеної транзакції – f885d6d737f6 9caf11038a8f 4771893d2 2d9f678e9cd2 99ec3bf4d686	Правильно

В даному розділі було обрано та аргументовано підходи до тестування системи. Було вирішено протестувати систему на різних рівнях за допомогою модульного, інтеграційного та навантажувального тестування. Були обрані інструменти для реалізації даних автоматизованих тестів. В результаті проведених тестів на різних рівнях системи та різними підходами виявлено, що функціонал реалізований у відповідності з вимогами до ПЗ, і система є повністю працездатною та здатною витримувати навантаження у вигляді використання системи типовими користувачами.

ВИСНОВКИ

У розділі аналізу предметної області та постановки задачі було проведено аналіз та огляд технології блокчейн, проаналізовано її основні реалізації та особливості, були проаналізовані сучасні види протоколів блокчейну. Було проведено аналіз криптовалюти Tronix, її специфіки та основні відмінності й переваги від інших криптовалют. Були встановлені типи та розглянуті різні види електронних гаманців, проведений аналіз існуючих рішень та проблем, огляд можливостей Telegram як платформи для розробки ботів. В результаті проведеного аналізу були визначені функціональні вимоги до розроблюваного програмного забезпечення, побудовані варіанти використання системи.

У розділі проектування програмного забезпечення ми визначили, що наша система буде побудована на основі клієнт-серверної архітектури. Було проведено аналіз архітектурних підходів для побудови серверної частини системи, в результаті чого була обрана мікросервісна архітектура, враховуючи її переваги в зрозумілості компонентів коду перед монолітним підходом. Були виділені окремі мікросервіси та визначена їх взаємодія при виконанні необхідного функціоналу. Був проведений аналіз типів баз даних та обрана нереляційна БД MongoDB, у зв'язку з зручністю її використання. Було визначено сутності для БД та їх структура. Був спроектований користувацький інтерфейс, визначивши поняття головного та вкладеного меню у вигляді відповідних клавіатур. Для побудови системи було вирішено використовувати мову програмування Java з фреймворком Spring та його модулями, які дозволяють швидко та зручно будувати серверні додатки , а також систему автоматичного збирання Gradle для обробки залежностей проекту.

У розділі програмної реалізації було проведене проектування окремих мікросервісів, визначені основні модулі кожного з них та побудована їх взаємодія. Було обрано додаткові інструменти для розробки, реалізовані програмні модулі, описані їх особливості за допомогою фрагментів коду. Було

створено інструкцію користувача по роботі з гаманцем, визначені вимоги до технічних та програмних засобів для запуску системи, а також описаний процес розгортання та встановлення системи.

У розділі тестування програмної системи були обрані методи тестування системи, створені тестові сценарії та проведений аналіз результатів тестування. В результаті проведених тестів на різних рівнях системи та різними підходами виявлено, що функціонал реалізований у відповідності з вимогами до ПЗ, і система є повністю працездатною та здатною витримувати навантаження у вигляді типових користувачів.

Використовуючи дану систему користувачі отримують перевагу у вигляді єдиного електронного гаманця на всіх системах, що дозволяє не встановлювати різні додатки для виконання одних й тих самих функцій, звільняє від необхідності освоєння різноманітних інтерфейсів програм. Користувачі можуть отримувати сповіщення про вхідні перекази та переглядати актуальний курс валюти – в багатьох інших існуючих рішеннях наведений функціонал не є реалізованим.

Серед можливих напрямів продовження роботи над проектом варто відмітити наступні варіанти: система обміну валютами, система для торгівлі валютами, реалізація платіжної системи тощо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Poelstra Andrew. On Stake and Consensus [Online] / A. Poelstra // Satoshi Nakamoto Institute. – Available: https://nakamotoinstitute.org/static/docs/on-stake-and-consensus.pdf

2. Sandeep Soni. Crypto market cap doubles to eye-popping $2.3 trillion in 3 months as mainstream adoption gets nearer [Online] / S. Soni // Financial Express. – Available: https://www.financialexpress.com/market/crypto-market-cap-doubles-to-eye-popping-2-3-trillion-in-3-months-as-mainstream-adoption-gets-nearer/2245246/

3. George Georgiev. Crypto Market Cap Added $300B in 7 Days as Altcoins Explode: The Weekly Recap [Online] / G. Georgiev // CryptoPotato. – Available: https://cryptopotato.com/crypto-market-cap-added-300b-in-7-days-as-altcoins-explode-the-weekly-recap/

4. Tron Whitepaper [Online] / Tron Network. – Available: https://tron.network/static/doc/white_paper_v_2_0.pdf

5. Andy Greenberg. Crypto Currency [Online] / A. Greenberg // Forbes. – Available: https://www.forbes.com/forbes/2011/0509/technology-psilocybin-bitcoins-gavin-andresen-crypto-currency.html?sh=6b3440ad353e

6. The great chain of being sure about things [Online] / The Economist. – Available: https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things

7. Ofir Beigel. What is Proof of Work [Online] / O. Beigel // 99 Bitcoins. – Available: https://99bitcoins.com/proof-of-work-proof-of-stake/

8. Reuben Jackson. Binance Suspends Ethereum and ERC-20 Token Withdrawals Before Quickly Reversing Course [Online] / R. Jackson // Bitcoin news. – Available: https://news.bitcoin.com/binance-suspends-ethereum-and-erc-20-token-withdrawals-before-quickly-reversing-course/

9. Что такое Delegated Proof Of Stake, и при чем здесь стейкинг? [Электронный ресурс] / Forklog. – Режим доступа: https://forklog.com/chto-takoe-delegated-proof-of-stake-i-pri-chem-zdes-stejking/

10. Li, Huawei; Li, Zhihuai; Tian, Na. Resource Bottleneck Analysis of the Blockchain Based on Tron's TPS. / L. Huawei; L. Zhihuai; T. Na. // Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery. Advances in Intelligent Systems and Computing. Springer International Publishing, 2020. – P. 944-950.

11. Tron resource model [Online] / The TRON Developer Hub. – Available: https://developers.tron.network/docs/resource-model

12. Hot Wallets vs. Cold Wallets [Online] / Cryptopedia. – Available: https://www.gemini.com/cryptopedia/crypto-wallets-hot-cold

13. MTProto Mobile Protocol [Online] / Telegram. – Available: https://core.telegram.org/mtproto

14. Mansoor Iqbal. Telegram Revenue and Usage Statistics [Online] / M. Iqbal // Business of Apps. – Available: https://www.businessofapps.com/data/telegram-statistics

15. Chris Richardson. Microservices Patterns: With examples in Java, 1 edition / C. Richardson. – Manning Publications. – 2018. – 520 p.

16. Roy Fielding. Architectural Styles and the Design of Network-based Software Architectures [Online] / R. Fielding // UNIVERSITY OF CALIFORNIA, IRVINE. – Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

ДОДАТОК А

(Обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту електронного гаманця для криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота.

2 Призначення розробки

Електронний гаманець для криптовалюти Tronix призначений для зберігання та переказів криптовалюти Tronix, перегляду історії переказів гаманця та інформації про інших учасників системи.

Користувачами програми є власники криптовалюти Tronix, та зацікавлені інформацією про систему особи.

Користувач може створити новий гаманець/акаунт в системі криптовалюти, або використовувати вже створений раніше гаманець, виконавши його імпорт в розроблюваний електронний гаманець. Використання балансу гаманця відбувається на підставі даних, збережених у блокчейні (базі даних) криптовалюти Tronix. Ці дані містять інформацію про актуальні баланси користувачів в системі, повну історію всіх переказів у системі.

Користувач може переказати кошти, які є на балансі його гаманця, та повністю переглянути історію переказів свого гаманця, знайти транзакцію, яка існує в системі або баланси інших користувачів.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Електронний гаманець криптовалюти Tronix повинний забезпечити виконання функцій, вказаних нижче.

Створення нового гаманця. Якщо у користувача відсутній активний гаманець, він може створити новий. Під час створення гаманця, користувач повинен створити пароль для гаманця, який буде використовуватись при виконанні інших дій,

пов’язаних з безпекою даних. Після створення паролю, користувачу відображається приватний ключ від нового створеного гаманця, який він повинен зберегти в надійному місці, після чого повторити його, ввівши у відповідне поле. Гаманець не буде активований у системі доти, доки на його рахунок не будуть переведені кошти, які його активують.

Імпорт гаманця. Якщо користувач вже має власний гаманець, створений раніше, він може використовувати його як активний, замість створення нового гаманця. Для цього користувач повинен ввести приватний ключ від свого гаманця, після чого він буде використовуватись при наступних операціях з гаманцем. Як і під час створення нового гаманця, користувач повинен буде створити для нього пароль.

Експорт гаманця. Користувач може побачити свій приватний ключ, після того як введе пароль, створений під час створення або імпорту гаманця. Також, користувач може вийти зі свого акаунту, відібравши в розроблюваного ПЗ доступ до нього.

Перегляд балансу гаманців. Користувач може переглянути баланс інших учасників блокчейну, ввівши в поле пошуку їх адресу. Також, якщо користувач має активний гаманець, він може переглядати свій баланс.

Функція пошуку транзакцій за її ідентифікатором. Користувач може переглянути інформацію про переказ коштів, ввівши ідентифікатор транзакції в поле пошуку. Ця інформація містить дату переказу, хто та кому переслав кошти, суму переказу, ідентифікатор транзакції, номер блоку, комісію, статус транзакції.

Перегляд курсу Tronix до інших валют, балансу в еквіваленті інших валют. Користувач може побачити курс та еквівалент власного балансу Tronix до гривні та долара США.

Застава Tronix для отримання ресурсів. Користувач може внести в заставу частину балансу, щоб отримати інші ресурси блокчейну (бали пропускної здатності, які необхідні для оплати переказів). Для цього користувач повинен вказати суму, яку бажає внести в заставу, та підтвердити дію. Перед виконанням операції, користувач бачить, скільки ресурсів отримає за заставу коштів.

Створення транзакцій. Користувач може переказати кошти на іншу адресу. Для цього він повинен вказати адресу для переказу та суму, яку бажає перевести. Перед

виконанням операції, користувач бачить, скільки ресурсів буде витрачено на виконання транзакції.

Перегляд історії транзакцій. Якщо користувач має активний гаманець, він може переглянути всі транзакції, які виконувались з цим гаманцем.

Отримання сповіщень про поповнення балансу. Якщо користувач має активний гаманець, він отримує сповіщення у вигляді повідомлення, коли на його баланс були зачислені кошти.

3.2 Вимоги до надійності

Розроблюване ПЗ повинно мати:

– можливість самовідновлення після збоїв, таких як відключення електроживлення тощо;

– обмеження максимальної кількості дій користувача в певний проміжок часу з метою захисту від спроб атак, які хочуть перенавантажити ПЗ;

– критична інформація в системі (паролі, приватні ключі тощо) повинні зберігатись у хешованому/шифрованому вигляді;

– помилки при роботі з зовнішніми сервісами, вхідними даними користувача повинні належним чином оброблятись, без загрози для стану ПЗ.

3.3 Вимоги до складу та параметрів технічних засобів

Системні вимоги для роботи ПЗ повинні бути наступними: тактова частота процесора – 2 МГц, кількість ядер – 2; обсяг оперативної пам’яті – 4 ГБ; обсяг вільного дискового простору – 500 Гб; підключення до інтернету зі швидкістю 100 Мбіт/с. Для роботи з ПЗ з користувацької сторони, повинен бути встановлений браузер або додаток Telegram.

3.4 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати на Linux Ubuntu 18.0 або новіших версіях вказаної ОС.

3.5 Вимоги до транспортування та зберігання

Програма, її документація поставляються у цифровому вигляді. Умови експлуатації програмного забезпечення збігаються з умовами експлуатації серверу, на якому буде розміщене ПЗ.

3.6 Спеціальні вимоги

Програма повинна мати дружній інтерфейс, розрахований на користувача середньої кваліфікації (з точки зору комп'ютерної грамотності). Мова програмування визначається вибором виконавця.

4. Вимоги до програмної документації

В ході розробки програми повинні бути підготовлені: текст програми, опис програми, інструкція з запуску та зупинки ПЗ, програма і методика випробувань, керівництво користувача.

5. Стадії та етапи розробки

Стадії та етапи розробки програмного забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.21 – 31.01.21	Обґрунтування необхідності розробки програми	Характеристика та опис ПЗ; підстави для розробки і призначення ПЗ; функціональні вимоги до розроблюваної системи; порядок контролю і приймання ПЗ.
Ескізний проект 01.02.21 – 14.02.21	Розробка ескізного проекту	Визначення структури вхідних і вихідних даних; попередній вибір технологій; визначення потрібних алгоритмів.
Технічний проект 15.02.21 – 28.02.21	Розробка технічного проекту	Затвердження структури вхідних і вихідних даних; розробка алгоритмів; розробка структури програми; вибір технологій.

Кінець таблиці А.1

1	2	3
Робочий проект 01.03.21 – 10.04.21	Розробка програмного забезпечення	Реалізація програмного забезпечення; виправлення помилок; тестування.
Розробка програмної документації 11.04.21 – 20.04.21	Розробка документації для програмного забезпечення	Розробка документації користувача, інструкції по запуску та зупинці ПЗ
Тестування системи 21.04.21 – 30.04.21	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Здача проекту	Здача програми замовнику	Передача вихідного коду та документації замовнику

6. Порядок контролю та приймання

Контроль і приймання розробки здійснюються на основі розробленої методики випробувань. При цьому перевіряється виконання всіх функцій програми групою користувачів та QA-спеціалістів. Прийом ПЗ замовником здійснюється після успішного тестування.

ДОДАТОК Б

(Обов’язковий)

ДІАГРАМИ КЛАСІВ ПРОГРАМИ

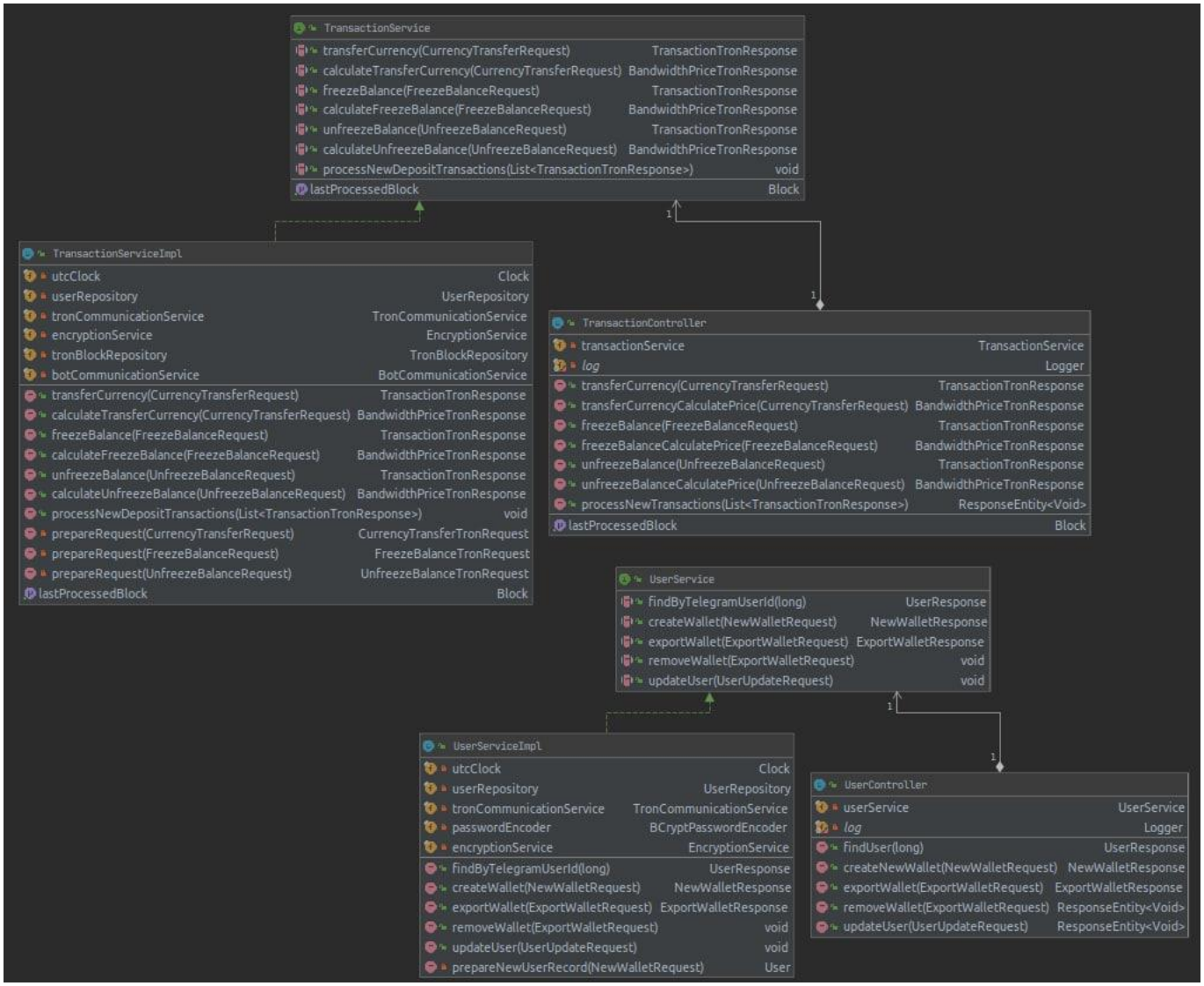


Рисунок Б.1 – Діаграма класів контролерів та сервісів be-core

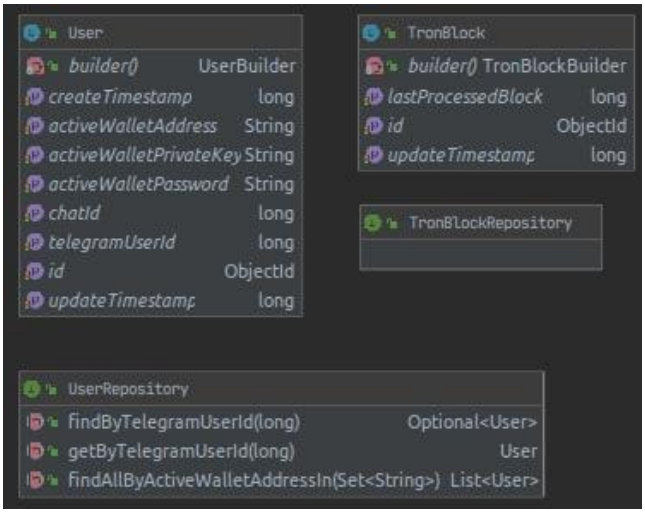


Рисунок Б.2 – Діаграма класів репозиторіїв та об’єктів бази даних be-core

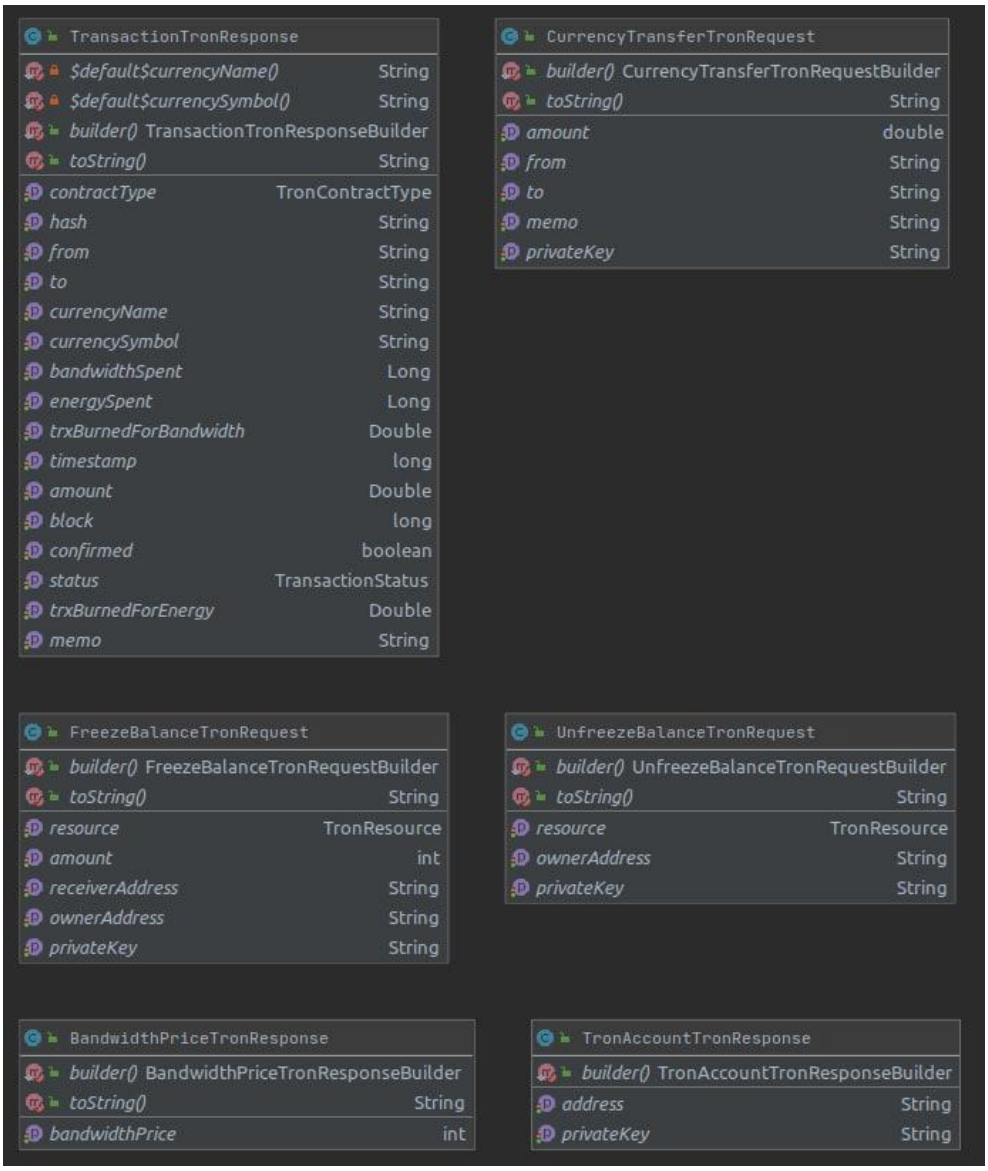


Рисунок Б.3 – Діаграма класів DTO об’єктів be-core для комунікації з be-tron

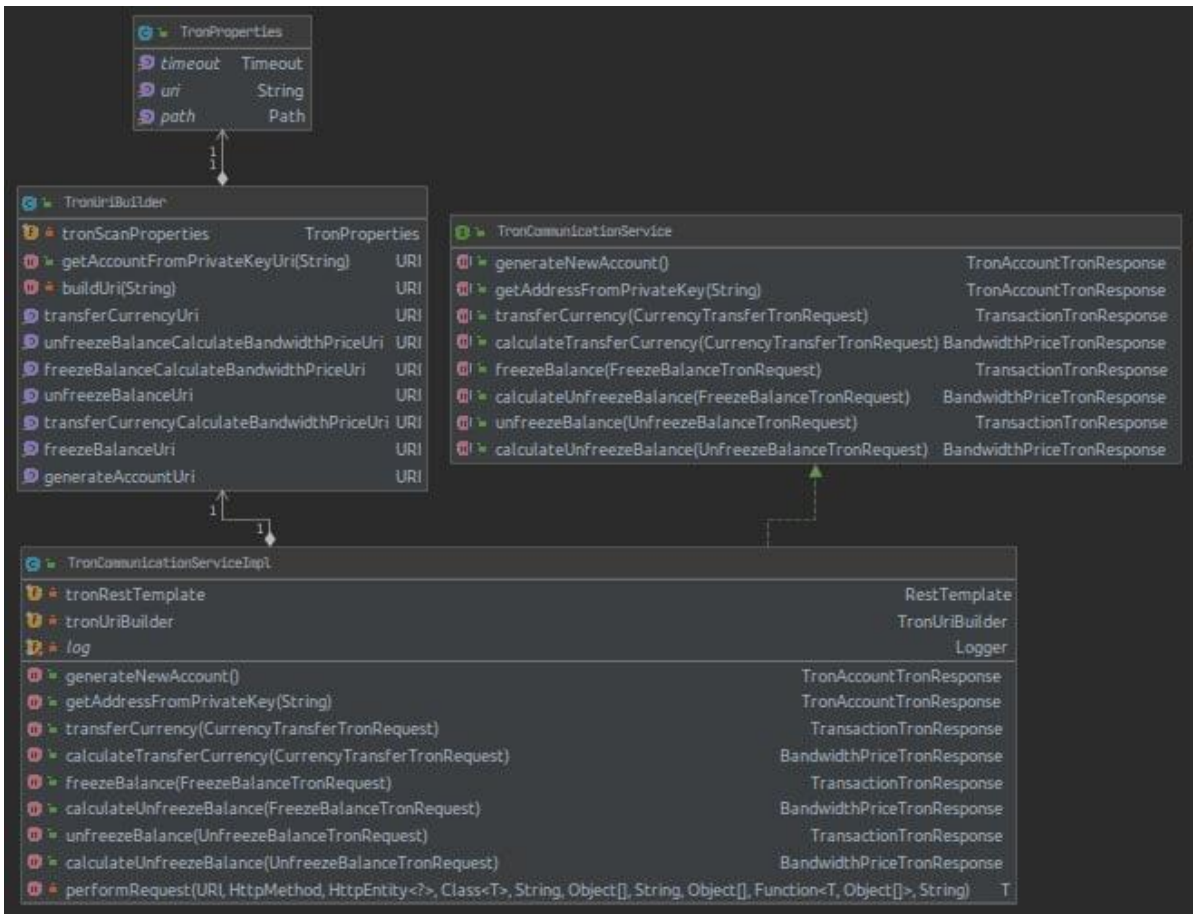


Рисунок Б.4 – Діаграма класів be-core для комунікації з be-tron

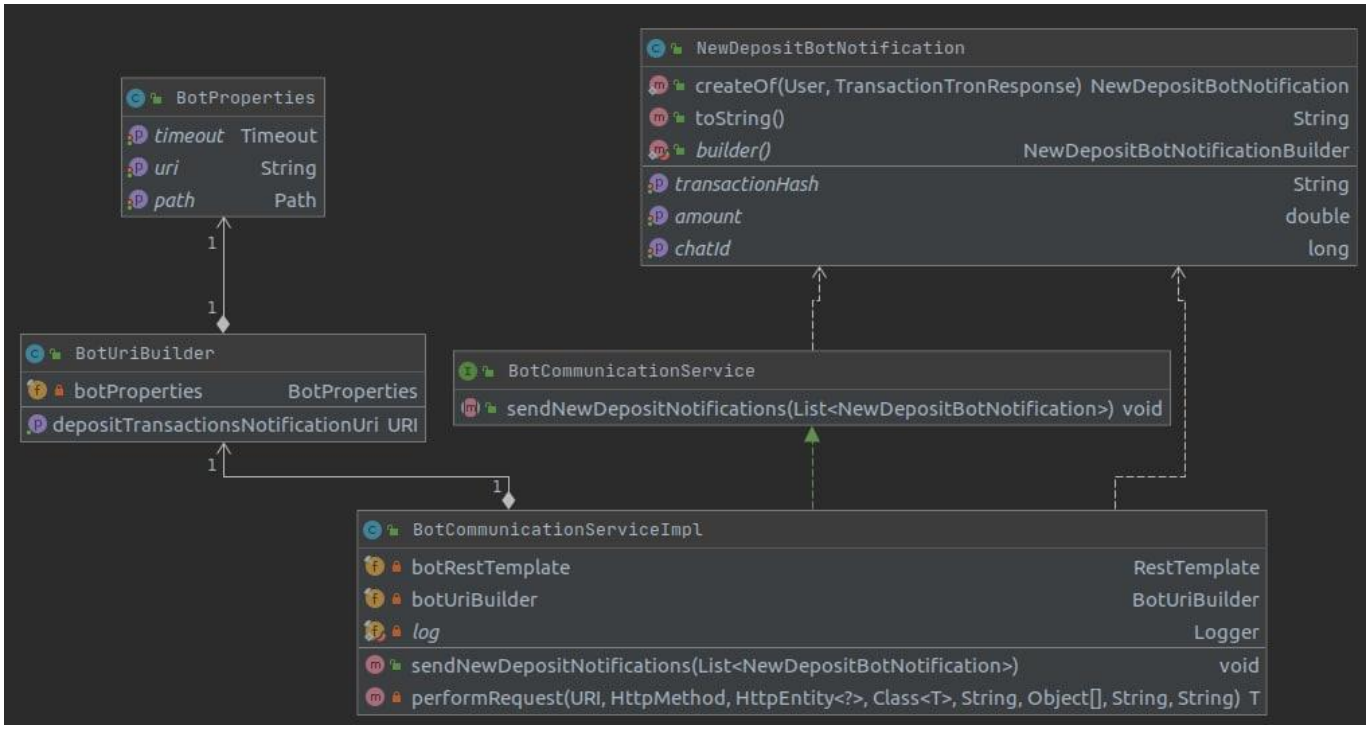


Рисунок Б.5 – Діаграма класів be-core для комунікації з be-bot

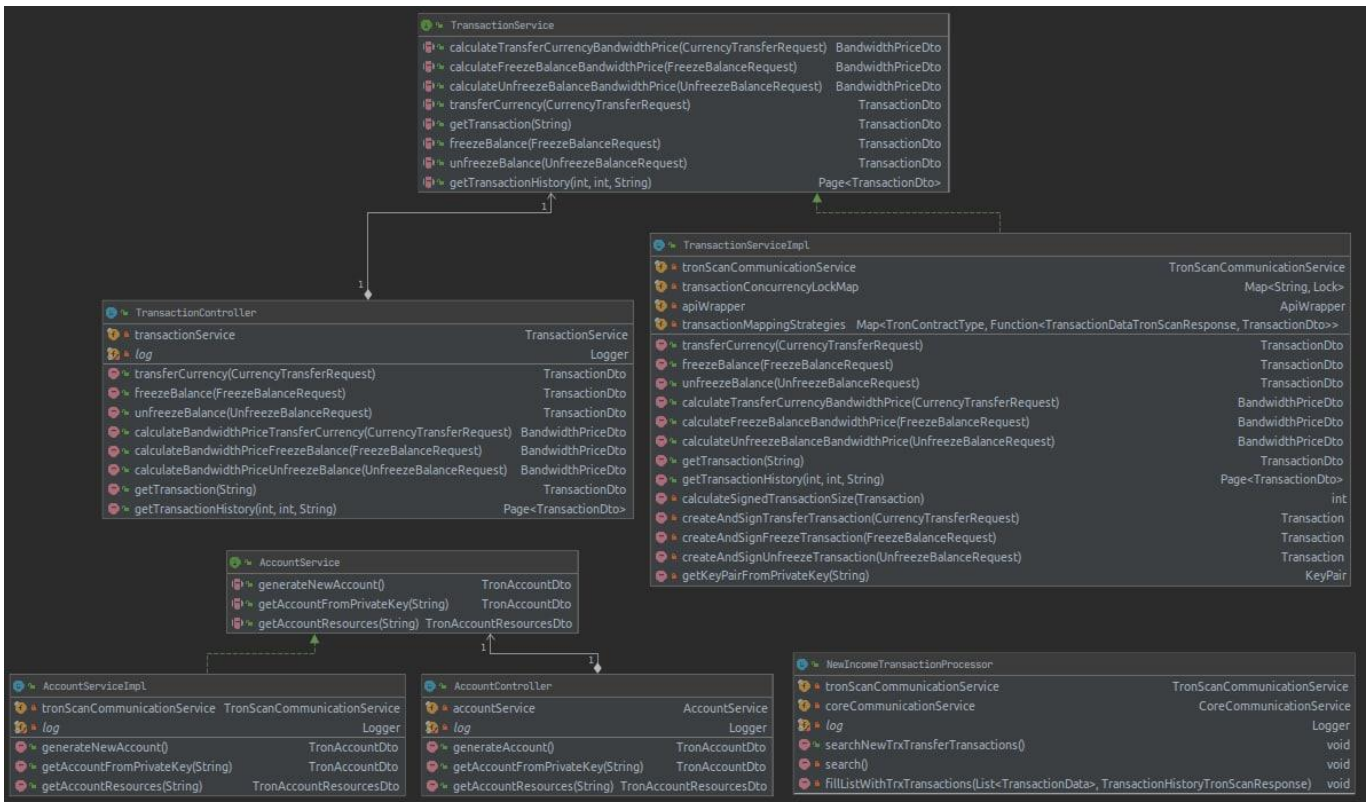


Рисунок Б.6 – Діаграма класів контролерів та сервісів be-tron

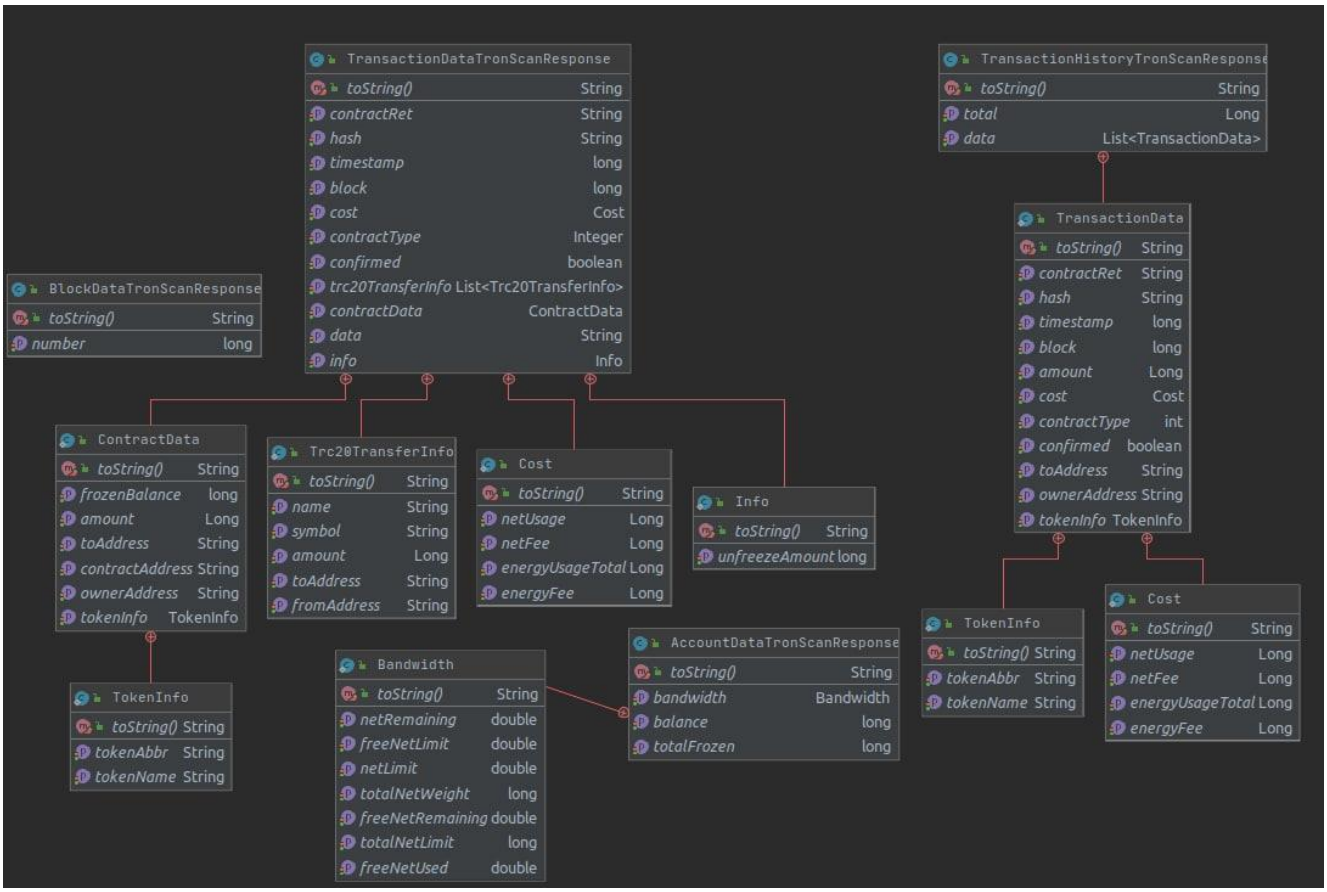


Рисунок Б.7 – Діаграма класів DTO об’єктів be-tron для комунікації з TronScan

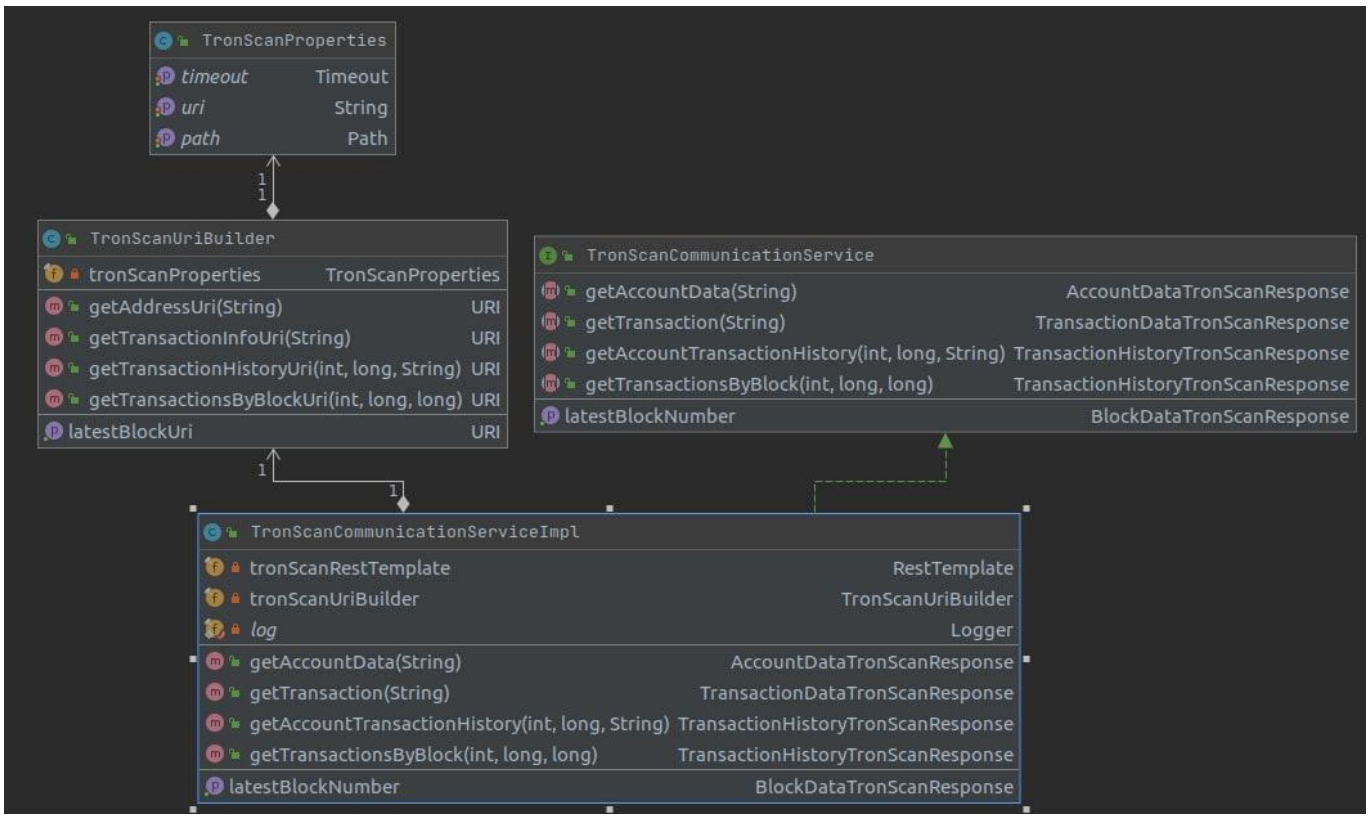


Рисунок Б.8 – Діаграма класів be-tron для комунікації з TronScan

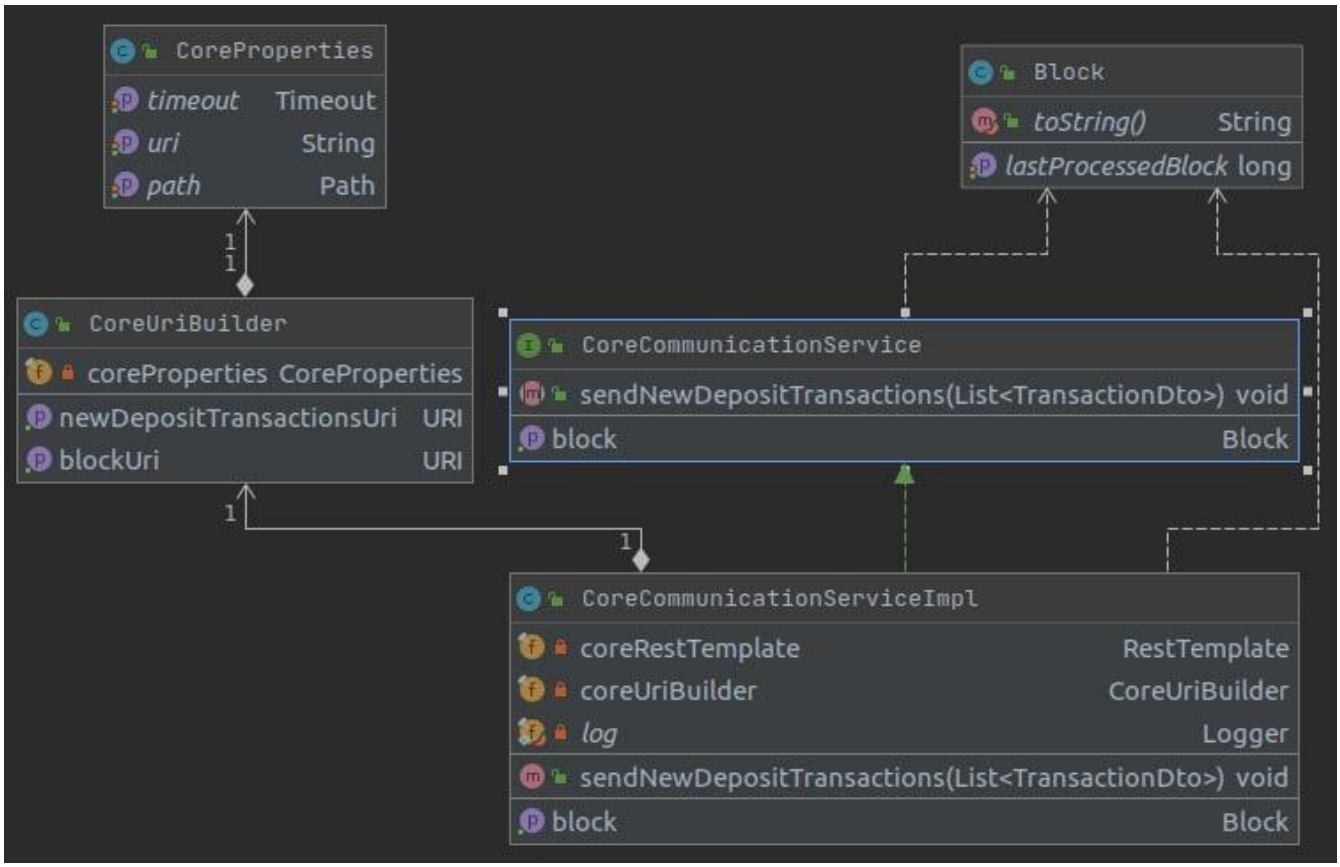


Рисунок Б.9 – Діаграма класів be-tron для комунікації з be-core

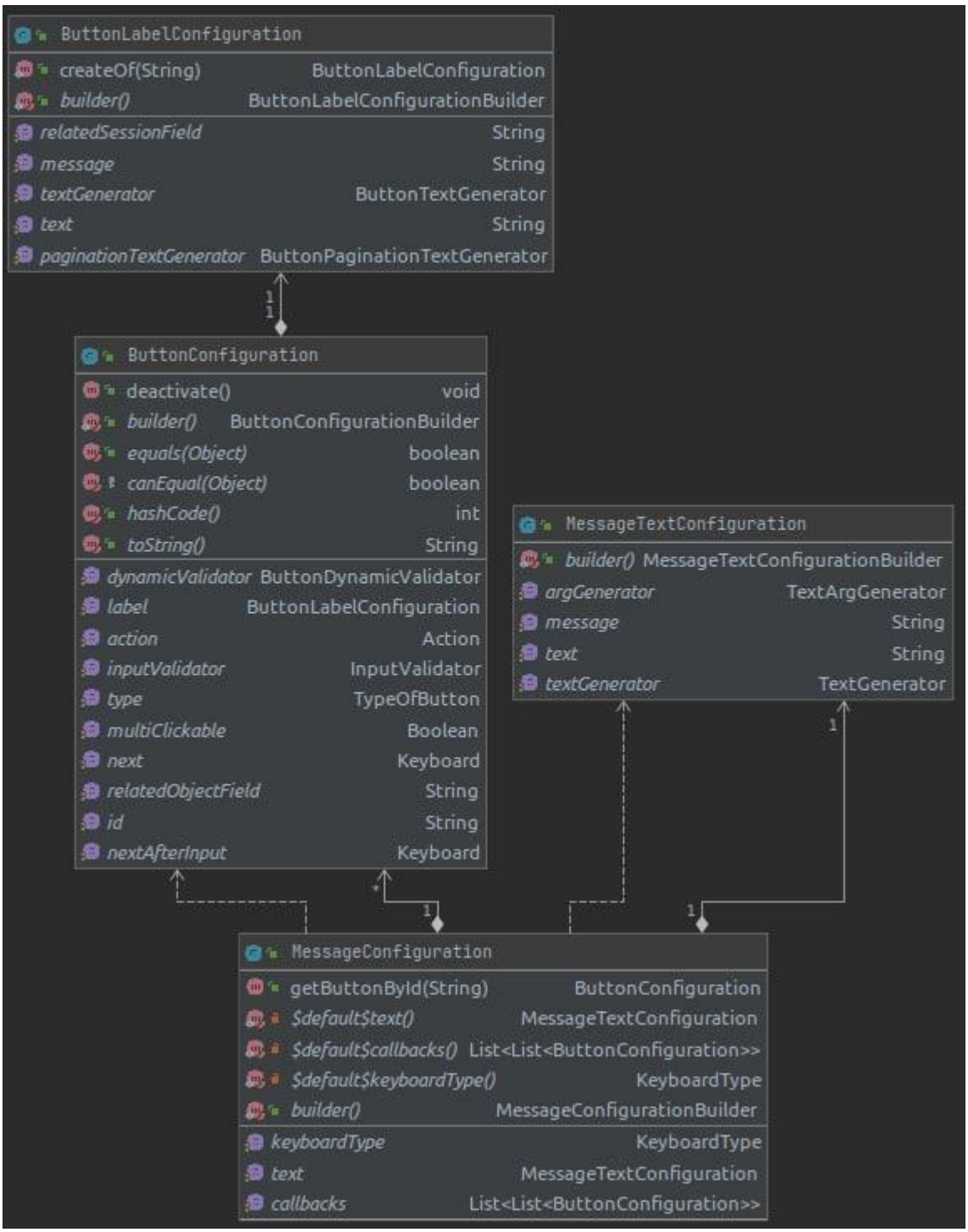


Рисунок Б.10 – Діаграма класів be-bot об’єктів конфігурацій повідомлень-інтерфейсу

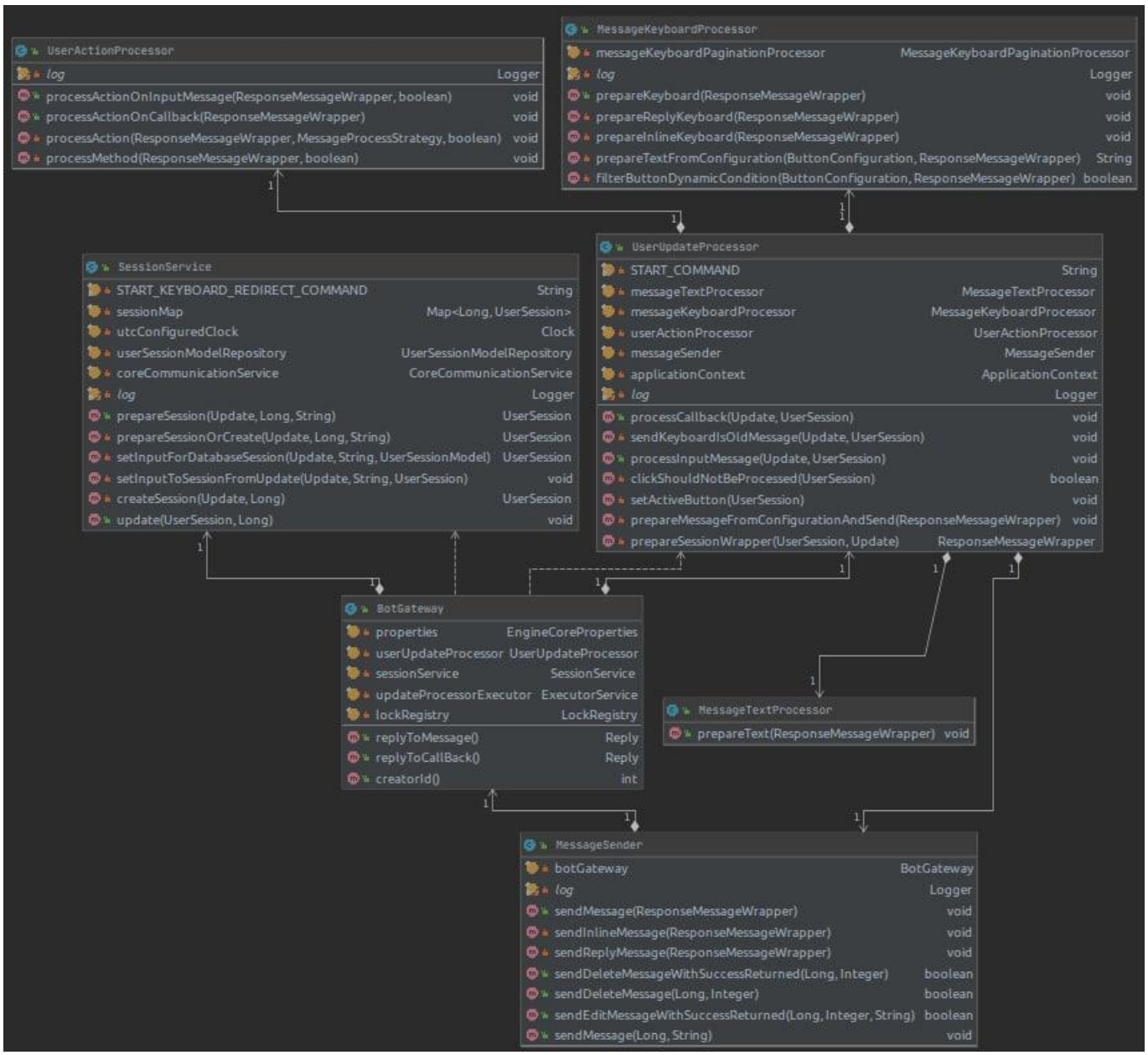


Рисунок Б.11 – Діаграма класів be-bot для обробки повідомлень користувача

ДОДАТОК В

(Обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

В.1 Програмний код мікросервіса be-core

Клас CoreProperties

```
package edu.diploma.ms.core.configuration.properties;
import lombok.Getter; import lombok.Setter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
@Getter @Setter @Component
public class CoreProperties {
@Value("${private-key-encryption-secret-key}") private String encryptionKey;}
```

Клас CoreConfiguration

```
package edu.diploma.ms.core.configuration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import java.time.Clock;
@Configuration
public class CoreConfiguration
{@Bean public Clock utcClock() {return Clock.systemUTC();}}
```

Клас RestTemplateConfiguration

```
package edu.diploma.ms.core.configuration;
import edu.diploma.ms.core.exception.InternalMicroserviceErrorHandler;
import edu.diploma.ms.core.exception.ServiceName;
import edu.diploma.ms.core.integration.bot.BotProperties;
import edu.diploma.ms.core.integration.tron.TronProperties;
import lombok.Getter; import lombok.Setter;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
import java.time.Duration;
@Getter @Setter @Configuration
public class RestTemplateConfiguration {
    @Bean
    public RestTemplate tronRestTemplate(RestTemplateBuilder builder, TronProperties
tronScanProperties) { return builder.rootUri(tronScanProperties.getUri())
                .errorHandler(new InternalMicroserviceErrorHandler(ServiceName.TRON))
.setConnectTimeout(Duration.ofMillis(tronScanProperties.getTimeout().getConnect()))
.setReadTimeout(Duration.ofMillis(tronScanProperties.getTimeout().getRead()))
```

```
                .build();}
    @Bean
    public RestTemplate botRestTemplate(RestTemplateBuilder builder, BotProperties
botProperties) {
        return builder.rootUri(botProperties.getUri()).errorHandler(new
InternalMicroserviceErrorHandler(ServiceName.BOT))
.setConnectTimeout(Duration.ofMillis(botProperties.getTimeout().getConnect()))
.setReadTimeout(Duration.ofMillis(botProperties.getTimeout().getRead())).build();}}
```

Клас TransactionStatus

```
package edu.diploma.ms.core.constants;
public enum TransactionStatus {SUCCESS, OUT_OF_ENERGY, UNKNOWN;}package
```

Клас TronContractType

```
edu.diploma.ms.core.constants;
public enum TronContractType {TRANSFER_CONTRACT, TRANSFER_ASSET_CONTRACT,
FREEZE_BALANCE_CONTRACT, UNFREEZE_BALANCE_CONTRACT, TRIGGER_SMART_CONTRACT, UNKNOWN;}
```

Клас TronResource

```
package edu.diploma.ms.core.constants;
import lombok.Getter; import lombok.RequiredArgsConstructor;
@Getter @RequiredArgsConstructor
public enum TronResource { BANDWIDTH(0), ENERGY(1); private final int code;}
```

Клас TronBlock

```
package edu.diploma.ms.core.db.model;
import lombok.*; import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
@Document(collection = "tronBlock")
public class TronBlock { @Id private ObjectId id; private long lastProcessedBlock;
private long updateTimestamp;}
```

Клас User

```
package edu.diploma.ms.core.db.model;
import lombok.*;
import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
```

```
@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
@Document(collection = "user")
public class User {@Id private ObjectId id; @Indexed(unique = true) private long
telegramUserId; private long chatId; private String activeWalletAddress;
private String activeWalletPassword; private String activeWalletPrivateKey;
private long createTimestamp; private long updateTimestamp;}
```

Клас TronBlockRepository

```
package edu.diploma.ms.core.db.repository;
import edu.diploma.ms.core.db.model.TronBlock; import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface TronBlockRepository extends MongoRepository<TronBlock, ObjectId> {}
```

Клас UserRepository

```
package edu.diploma.ms.core.db.repository;

import edu.diploma.ms.core.db.model.User; import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import java.util.List; import java.util.Optional; import java.util.Set;

@Repository
public interface UserRepository extends MongoRepository<User, ObjectId> {
    Optional<User> findByTelegramUserId(long telegramUserId);
    User getByTelegramUserId(long telegramUserId);
    List<User> findAllByActiveWalletAddressIn(Set<String> addressesToFind);}
```

Клас BaseRuntimeException

```
package edu.diploma.ms.core.exception;
import lombok.Getter;
public class BaseRuntimeException extends RuntimeException {
@Getter private final ExceptionCode exceptionCode;

public BaseRuntimeException(ExceptionCode exceptionCode) {
        super(exceptionCode.getMessage()); this.exceptionCode = exceptionCode; }

BaseRuntimeException(ExceptionCode exceptionCode, String message, Throwable
throwable) { super(message, throwable); this.exceptionCode = exceptionCode; }

BaseRuntimeException(ExceptionCode exceptionCode, String message) {
        super(message); this.exceptionCode = exceptionCode;}

@Override public String getMessage() {return super.getMessage();}}
```

Клас ErrorInfo

```
package edu.diploma.ms.core.exception;
import lombok.*;
import java.util.List;
@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
public class ErrorInfo {private List<String> messages; private Integer code;}
```

Клас ExceptionCode

```
package edu.diploma.ms.core.exception;
import lombok.Getter; import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;

@Getter @RequiredArgsConstructor
public enum ExceptionCode {
SOME_PARAMETERS_ABSENT_OR_INVALID_EXCEPTION(4000102, "Some parameters are absent, or
invalid.", HttpStatus.BAD_REQUEST),
WRONG_PASSWORD_EXCEPTION(4000202, "Wrong password", HttpStatus.BAD_REQUEST),

INTERNAL_SERVICE_EXCEPTION(5000102, "Issue with internal service",
HttpStatus.INTERNAL_SERVER_ERROR),
UNKNOWN_EXCEPTION(9999902, "Unknown error", HttpStatus.INTERNAL_SERVER_ERROR);
private final int errorCode;
private final String message; private final HttpStatus status;}
```

Клас InternalMicroserviceErrorHandler

```
package edu.diploma.ms.core.exception;
import lombok.AllArgsConstructor; import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest; import java.util.Collections;
@Slf4j @AllArgsConstructor @ControllerAdvice
public class HttpRequestRuntimeExceptionProcessor {
private static final String ERROR_MESSAGE_FORMAT = " | Message: ";
@ExceptionHandler(BaseRuntimeException.class)
public ResponseEntity<ErrorInfo> handleBaseException(BaseRuntimeException e,
HttpServletRequest request) { log(request, e); ErrorInfo errorInfo =
createErrorDto(e);
return new ResponseEntity<>(errorInfo, e.getExceptionCode().getStatus());}

@ExceptionHandler(RuntimeException.class)
public ResponseEntity<ErrorInfo> handleRuntimeException(RuntimeException e,
HttpServletRequest request) { logError(request, e);
ErrorInfo errorInfo = createErrorDto(ExceptionCode.UNKNOWN_EXCEPTION,
ExceptionCode.UNKNOWN_EXCEPTION.getMessage());
return new ResponseEntity<>(errorInfo, HttpStatus.INTERNAL_SERVER_ERROR); }

@ExceptionHandler(RemoteMsException.class)
public ResponseEntity<ErrorInfo> handleRemoteMsPassThroughException(RemoteMsException
e, HttpServletRequest request) {
```

```
logWarning(request, e.getMessage());
ErrorInfo errorInfoDto = e.getErrorInfoDto();
return new ResponseEntity<>(errorInfoDto, HttpStatus.BAD_REQUEST); }
private void log(HttpServletRequest request, BaseRuntimeException
baseRuntimeException) { String message = request.getRequestURI() + "?" +
request.getQueryString() + ERROR_MESSAGE_FORMAT + baseRuntimeException;
if (baseRuntimeException.getExceptionCode().getStatus() ==
HttpStatus.INTERNAL_SERVER_ERROR) {log.error(message);} else {log.warn(message);}}

private static ErrorInfo createErrorDto(BaseRuntimeException baseRuntimeException) {
ExceptionCode exceptionCode = baseRuntimeException.getExceptionCode(); ErrorInfo
errorInfo = new ErrorInfo(); errorInfo.setCode(exceptionCode.getErrorCode());
errorInfo.setMessages(Collections.singletonList(exceptionCode.getMessage()));
return errorInfo;}

    private static ErrorInfo createErrorDto(ExceptionCode exceptionCode, String
message) {ErrorInfo errorInfo = new ErrorInfo();
errorInfo.setCode(exceptionCode.getErrorCode());
errorInfo.setMessages(Collections.singletonList(message)); return errorInfo;}
private void logError(HttpServletRequest request, RuntimeException exception) {
String message = request.getRequestURI() + "?" + request.getQueryString();
log.error(message, exception); }
private void logError(HttpServletRequest request, Exception exception) {
String message = request.getRequestURI() + "?" + request.getQueryString();
log.error(message, exception); }
private void logWarning(HttpServletRequest request, String errorMessage) {
String message = request.getRequestURI() + "?" + request.getQueryString() +
ERROR_MESSAGE_FORMAT + errorMessage; log.warn(message);}}
```

Клас BotCommunicationService

```
package edu.diploma.ms.core.integration.bot.communication;
import edu.diploma.ms.core.integration.bot.model.NewDepositBotNotification;
import java.util.List;
public interface BotCommunicationService {
void sendNewDepositNotifications(List<NewDepositBotNotification> notifications);}
```

Клас BotCommunicationServiceImpl

```
package edu.diploma.ms.core.integration.bot.communication;
import edu.diploma.ms.core.exception.RemoteMsAccessRuntimeException;
import edu.diploma.ms.core.integration.bot.BotUriBuilder;
import edu.diploma.ms.core.integration.bot.model.NewDepositBotNotification;
import lombok.RequiredArgsConstructor; import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpMethod;
import org.springframework.stereotype.Service;
import org.springframework.web.client.ResourceAccessException;
import org.springframework.web.client.RestTemplate;
import java.net.URI; import java.util.List;
@Slf4j @RequiredArgsConstructor @Service
public class BotCommunicationServiceImpl implements BotCommunicationService {
private final RestTemplate botRestTemplate;private final BotUriBuilder botUriBuilder;

public void sendNewDepositNotifications(List<NewDepositBotNotification>
notifications) { URI addressUri =
```

```
botUriBuilder.getDepositTransactionsNotificationUri(); performRequest(addressUri,
HttpMethod.POST, new HttpEntity<>(notifications), Void.class,
"Request to bot to send new deposit notifications: {}", new Object[]{notifications},
"Request to bot to send new deposit notifications processed.",
"Error when sending new deposit notifications: {}");}

private <T> T performRequest(URI uri, HttpMethod method, HttpEntity<?> body, Class<T>
responseType, String preLog, Object[] preLogArgs, String postLog, String errorLog) {
log.info(preLog, preLogArgs);
try {T responseBody = botRestTemplate.exchange(uri, method, body,
responseType).getBody(); log.info(postLog);
return responseBody;} catch (ResourceAccessException e) {log.error(errorLog,
e.getMessage());
throw new RemoteMsAccessRuntimeException("Error when connecting to tron: " +
e.getMessage());}}}
```

Клас BotProperties

```
package edu.diploma.ms.core.integration.bot;
import lombok.Getter; import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;
@Getter @Setter @Component @ConfigurationProperties(prefix = "ms.bot")
public class BotProperties {
private String uri; private Path path; private Timeout timeout;
@Getter @Setter
public static class Path { private String newDepositNotification; }
@Getter @Setter
public static class Timeout { private int connect; private int read; }}
```

Клас BotUriBuilder

```
edu.diploma.ms.core.integration.bot;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import org.springframework.web.util.UriComponentsBuilder;
import java.net.URI;
@RequiredArgsConstructor @Component
public class BotUriBuilder { private final BotProperties botProperties;
public URI getDepositTransactionsNotificationUri() { return
UriComponentsBuilder.fromUriString(botProperties.getUri()).path(botProperties.getPath
().getNewDepositNotification()).build().encode().toUri();}}
```

Клас NewDepositBotNotification

```
package edu.diploma.ms.core.integration.bot.model;
import edu.diploma.ms.core.db.model.User;
import edu.diploma.ms.core.integration.tron.model.TransactionTronResponse;
import lombok.AllArgsConstructor;
import lombok.Builder; import lombok.Getter;
import lombok.NoArgsConstructor; import lombok.Setter;

@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
public class NewDepositBotNotification {
```

```
private long chatId; private String transactionHash; private double amount;
public static NewDepositBotNotification createOf(User user, TransactionTronResponse
trx) { return new NewDepositBotNotification(user.getChatId(), trx.getHash(),
trx.getAmount());}}
```

Клас BandwidthPriceTronResponse

```
package edu.diploma.ms.core.integration.tron.model;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor
@AllArgsConstructor @JsonInclude(JsonInclude.Include.NON_NULL)
public class BandwidthPriceTronResponse { private int bandwidthPrice;}
```

Клас CurrencyTransferTronRequest

```
package edu.diploma.ms.core.integration.tron.model;
import lombok.*;
@Getter @Setter @Builder @ToString
@NoArgsConstructor @AllArgsConstructor
public class CurrencyTransferTronRequest {
private String from; @ToString.Exclude private String privateKey;
private String to; private double amount; private String memo;}
```

Клас FreezeBalanceTronRequest

```
package edu.diploma.ms.core.integration.tron.model;
import edu.diploma.ms.core.constants.TronResource;
import lombok.*;
@Getter @Setter @Builder @ToString
@NoArgsConstructor @AllArgsConstructor
public class FreezeBalanceTronRequest {
private String ownerAddress; @ToString.Exclude
private String privateKey; private TronResource resource;
private int amount; private String receiverAddress;}
```

Клас TransactionTronResponse

```
package edu.diploma.ms.core.integration.tron.model;
import com.fasterxml.jackson.annotation.JsonInclude;
import edu.diploma.ms.core.constants.TransactionStatus;
import edu.diploma.ms.core.constants.TronContractType;
import lombok.*;
@Getter @Setter @Builder @ToString
@NoArgsConstructor @AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class TransactionTronResponse {
private String hash; private long block;
private long timestamp; private boolean confirmed;
private TransactionStatus status;
```

```
private TronContractType contractType;
private String from; private String to;
@Builder.Default private String currencyName = "trx";
@Builder.Default private String currencySymbol = "trx";
private Double amount; private String memo;
private Double trxBurnedForBandwidth; private Double trxBurnedForEnergy;
private Long bandwidthSpent; private Long energySpent;}
```

Клас TronAccountTronResponse

```
package edu.diploma.ms.core.integration.tron.model;
import lombok.*;
@Getter @Setter @Builder
@NoArgsConstructor @AllArgsConstructor
public class TronAccountTronResponse {
private String privateKey;
private String address;}
```

Клас UnfreezeBalanceTronRequest

```
package edu.diploma.ms.core.integration.tron.model;
import edu.diploma.ms.core.constants.TronResource;
import lombok.*;

@Getter @Setter @Builder @ToString
@NoArgsConstructor @AllArgsConstructor
public class UnfreezeBalanceTronRequest {
private String ownerAddress;
@ToString.Exclude private String privateKey;
private TronResource resource;}
```

Клас TronProperties

```
package edu.diploma.ms.core.integration.tron;
import lombok.*;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Getter @Setter @Component
@ConfigurationProperties(prefix = "ms.tron")
public class TronProperties {
    private String uri; private Path path; private Timeout timeout;
    @Getter @Setter
    public static class Path {
        private String getAccountFromPrivateKey;
        private String generateAccount; private String transferCurrency;
        private String freezeBalance; private String unfreezeBalance;
        private String transferCurrencyCalculateBandwidthPrice;
        private String freezeBalanceCalculateBandwidthPrice;
        private String unfreezeBalanceCalculateBandwidthPrice;
    }
    @Getter @Setter
    public static class Timeout { private int connect; private int read; }}
```

Клас TronUriBuilder

```
package edu.diploma.ms.core.integration.tron;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import org.springframework.web.util.UriComponentsBuilder;
import java.net.URI;
@RequiredArgsConstructor @Component
public class TronUriBuilder {

    private final TronProperties tronScanProperties;
    public URI getGenerateAccountUri() {
        return buildUri(tronScanProperties.getPath().getGenerateAccount());}

    public URI getAccountFromPrivateKeyUri(String privateKey) {
        return UriComponentsBuilder.fromUriString(tronScanProperties.getUri())
                .path(tronScanProperties.getPath().getGetAccountFromPrivateKey())
                .queryParam("privateKey", privateKey).build().encode().toUri();}
    public URI getTransferCurrencyUri() {
        return buildUri(tronScanProperties.getPath().getTransferCurrency());}
    public URI getFreezeBalanceUri() {
        return buildUri(tronScanProperties.getPath().getFreezeBalance());}
    public URI getUnfreezeBalanceUri() {
        return buildUri(tronScanProperties.getPath().getUnfreezeBalance());}
    public URI getTransferCurrencyCalculateBandwidthPriceUri() {
        return
buildUri(tronScanProperties.getPath().getTransferCurrencyCalculateBandwidthPrice());}
    public URI getFreezeBalanceCalculateBandwidthPriceUri() {
        return
buildUri(tronScanProperties.getPath().getFreezeBalanceCalculateBandwidthPrice());}
    public URI getUnfreezeBalanceCalculateBandwidthPriceUri() {
        return
buildUri(tronScanProperties.getPath().getUnfreezeBalanceCalculateBandwidthPrice());}
    private URI buildUri(String path) {
        return UriComponentsBuilder.fromUriString(tronScanProperties.getUri())
                .path(path).build().encode().toUri();}}
```

Клас EncryptionService

```
package edu.diploma.ms.core.service;

public interface EncryptionService {

    String encrypt(String value);
    String decrypt(String value);
}
```

Клас EncryptionServiceImpl

```
package edu.diploma.ms.core.service;
import edu.diploma.ms.core.configuration.properties.CoreProperties;
import edu.diploma.ms.core.util.CryptoAES;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
```

```
@RequiredArgsConstructor @Service
public class EncryptionServiceImpl implements EncryptionService {
    private final CoreProperties coreProperties;
    @Override public String encrypt(String value) {
        byte[] bytes = CryptoAES.encryptWithRandomIv(value.getBytes(),
coreProperties.getEncryptionKey()); return new String(bytes);}
    @Override public String decrypt(String value) {
        byte[] bytes = CryptoAES.decryptWithRandomIv(value.getBytes(),
coreProperties.getEncryptionKey()); return new String(bytes);}}
```

Клас TransactionService

```
package edu.diploma.ms.core.service;
import edu.diploma.ms.core.integration.tron.model.BandwidthPriceTronResponse;
import edu.diploma.ms.core.integration.tron.model.TransactionTronResponse;
import edu.diploma.ms.core.web.model.Block;
import edu.diploma.ms.core.web.model.CurrencyTransferRequest;
import edu.diploma.ms.core.web.model.FreezeBalanceRequest;
import edu.diploma.ms.core.web.model.UnfreezeBalanceRequest;
import java.util.List;

public interface TransactionService {
    TransactionTronResponse transferCurrency(CurrencyTransferRequest
currencyTransferRequest);
    BandwidthPriceTronResponse calculateTransferCurrency(CurrencyTransferRequest
currencyTransferRequest);
    TransactionTronResponse freezeBalance(FreezeBalanceRequest request);
    BandwidthPriceTronResponse calculateFreezeBalance(FreezeBalanceRequest request);
    TransactionTronResponse unfreezeBalance(UnfreezeBalanceRequest request);
    BandwidthPriceTronResponse calculateUnfreezeBalance(UnfreezeBalanceRequest
request); Block getLastProcessedBlock();
    void processNewDepositTransactions(List<TransactionTronResponse>
depositTransactions);}
```

Клас TransactionServiceImpl

```
package edu.diploma.ms.core.service;
import edu.diploma.ms.core.constants.TronResource;
import edu.diploma.ms.core.db.model.TronBlock;
import edu.diploma.ms.core.db.model.User; import edu.diploma.ms.core.web.model.Block;
import edu.diploma.ms.core.db.repository.TronBlockRepository;
import edu.diploma.ms.core.db.repository.UserRepository;
import edu.diploma.ms.core.integration.bot.communication.BotCommunicationService;
import edu.diploma.ms.core.integration.bot.model.NewDepositBotNotification;
import edu.diploma.ms.core.integration.tron.communication.TronCommunicationService;
import edu.diploma.ms.core.integration.tron.model.*;
import edu.diploma.ms.core.web.model.CurrencyTransferRequest;
import edu.diploma.ms.core.web.model.FreezeBalanceRequest;
import edu.diploma.ms.core.web.model.UnfreezeBalanceRequest;
import lombok.RequiredArgsConstructor; import org.springframework.stereotype.Service;

import java.time.Clock; import java.util.*; import java.util.stream.Collectors;

@RequiredArgsConstructor @Service
public class TransactionServiceImpl implements TransactionService {

    private final Clock utcClock; private final UserRepository userRepository;
```

```
    private final TronCommunicationService tronCommunicationService;
    private final EncryptionService encryptionService;
    private final TronBlockRepository tronBlockRepository;
    private final BotCommunicationService botCommunicationService;

@Override public TransactionTronResponse transferCurrency(CurrencyTransferRequest
currencyTransferRequest) { CurrencyTransferTronRequest currencyTransferTronRequest =
prepareRequest(currencyTransferRequest); return
tronCommunicationService.transferCurrency(currencyTransferTronRequest); }

    @Override public BandwidthPriceTronResponse
calculateTransferCurrency(CurrencyTransferRequest currencyTransferRequest) {
        CurrencyTransferTronRequest currencyTransferTronRequest =
prepareRequest(currencyTransferRequest); return
tronCommunicationService.calculateTransferCurrency(currencyTransferTronRequest);}

    @Override public TransactionTronResponse freezeBalance(FreezeBalanceRequest
request) { FreezeBalanceTronRequest requestTron = prepareRequest(request);
        return tronCommunicationService.freezeBalance(requestTron); }

    @Override public BandwidthPriceTronResponse
calculateFreezeBalance(FreezeBalanceRequest request) { FreezeBalanceTronRequest
requestTron = prepareRequest(request); return
tronCommunicationService.calculateUnfreezeBalance(requestTron); }

    @Override public TransactionTronResponse unfreezeBalance(UnfreezeBalanceRequest
request) { UnfreezeBalanceTronRequest requestTron = prepareRequest(request);
        return tronCommunicationService.unfreezeBalance(requestTron); }

    @Override public BandwidthPriceTronResponse
calculateUnfreezeBalance(UnfreezeBalanceRequest request) {
        UnfreezeBalanceTronRequest requestTron = prepareRequest(request);
        return tronCommunicationService.calculateUnfreezeBalance(requestTron); }

    @Override public Block getLastProcessedBlock() { List<TronBlock> all =
tronBlockRepository.findAll(); if(all.size() == 1) { return new
Block(all.get(0).getLastProcessedBlock()); } else { return new Block(0); }}

@Override public void processNewDepositTransactions(List<TransactionTronResponse>
depositTransactions) {
long lastBlock = depositTransactions.get(0).getBlock();
Set<String> addressesToCheck =
depositTransactions.stream().map(TransactionTronResponse::getTo).collect(Collectors.t
oSet());
List<User> usersToNotify =
userRepository.findAllByActiveWalletAddressIn(addressesToCheck);
        Map<String, User> activeAddressToTelegramChatId = usersToNotify.stream()
.collect(Collectors.toMap(User::getActiveWalletAddress, user -> user));
        depositTransactions = depositTransactions.stream().filter(trx ->
activeAddressToTelegramChatId.containsKey(trx.getTo()))
.collect(Collectors.toList());
        List<NewDepositBotNotification> newDepositBotNotifications = new
ArrayList<>();
depositTransactions.forEach(trx -> { User user =
activeAddressToTelegramChatId.get(trx.getTo()); NewDepositBotNotification
notification = NewDepositBotNotification.createOf(user, trx);
newDepositBotNotifications.add(notification);});
        if(!newDepositBotNotifications.isEmpty()) {
botCommunicationService.sendNewDepositNotifications(newDepositBotNotifications); }

        List<TronBlock> all = tronBlockRepository.findAll();
        if (all.size() == 1) {
TronBlock tronBlock = all.get(0); tronBlock.setLastProcessedBlock(lastBlock);
tronBlock.setUpdateTimestamp(utcClock.millis());
```

```
tronBlockRepository.save(tronBlock); } else {   tronBlockRepository
.save(TronBlock.builder().lastProcessedBlock(lastBlock).build());}}

    private CurrencyTransferTronRequest prepareRequest(CurrencyTransferRequest
currencyTransferRequest) {
        User user =
userRepository.getByTelegramUserId(currencyTransferRequest.getTelegramUserId());
        String privateKey =
encryptionService.decrypt(user.getActiveWalletPrivateKey());
        return
CurrencyTransferTronRequest.builder().from(user.getActiveWalletAddress())
.amount(currencyTransferRequest.getAmount()).memo(currencyTransferRequest.getMemo())
.to(currencyTransferRequest.getTo()).privateKey(privateKey).build();}

    private FreezeBalanceTronRequest prepareRequest(FreezeBalanceRequest
freezeBalanceRequest) {
        User user =
userRepository.getByTelegramUserId(freezeBalanceRequest.getTelegramUserId());
        String privateKey =
encryptionService.decrypt(user.getActiveWalletPrivateKey());
        return FreezeBalanceTronRequest.builder()
.resource(TronResource.BANDWIDTH).ownerAddress(user.getActiveWalletAddress())
.privateKey(privateKey).amount(freezeBalanceRequest.getAmount()).build();}
private UnfreezeBalanceTronRequest prepareRequest(UnfreezeBalanceRequest
unfreezeBalanceRequest) {
        User user =
userRepository.getByTelegramUserId(unfreezeBalanceRequest.getTelegramUserId());
        String privateKey =
encryptionService.decrypt(user.getActiveWalletPrivateKey());
        return UnfreezeBalanceTronRequest.builder().resource(TronResource.BANDWIDTH)
.ownerAddress(user.getActiveWalletAddress()).privateKey(privateKey).build();}}
```

Клас UserService

```
package edu.diploma.ms.core.service;

import edu.diploma.ms.core.web.model.*;
public interface UserService {
    UserResponse findByTelegramUserId(long telegramUserId);

    NewWalletResponse createWallet(NewWalletRequest newWalletRequest);
    ExportWalletResponse exportWallet(ExportWalletRequest exportWalletRequest);

    void removeWallet(ExportWalletRequest exportWalletRequest);
    void updateUser(UserUpdateRequest userUpdateRequest);
}
```

Клас UserServiceImpl

```
package edu.diploma.ms.core.service;

import edu.diploma.ms.core.db.model.User;
import edu.diploma.ms.core.db.repository.UserRepository;
import edu.diploma.ms.core.exception.WrongPasswordRuntimeException;
import edu.diploma.ms.core.integration.tron.communication.TronCommunicationService;
import edu.diploma.ms.core.integration.tron.model.TronAccountTronResponse;
```

```
import edu.diploma.ms.core.web.model.*; import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service; import java.time.Clock;

@RequiredArgsConstructor @Service
public class UserServiceImpl implements UserService {

private final Clock utcClock;
private final UserRepository userRepository;
private final TronCommunicationService tronCommunicationService;
private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder(10);
private final EncryptionService encryptionService;

    @Override
    public UserResponse findByTelegramUserId(long telegramUserId) { return
UserResponse.createOf(userRepository.findByTelegramUserId(telegramUserId).orElse(User
.builder().build()));}

    @Override
    public NewWalletResponse createWallet(NewWalletRequest newWalletRequest) {
User user = userRepository.findByTelegramUserId(newWalletRequest.getTelegramUserId())
.orElse(prepareNewUserRecord(newWalletRequest));
        TronAccountTronResponse account;
if(newWalletRequest.getActiveWalletPrivateKey() != null) { account =
tronCommunicationService.getAddressFromPrivateKey(newWalletRequest.getActiveWalletPri
vateKey()); } else { account = tronCommunicationService.generateNewAccount();}

user.setActiveWalletAddress(account.getAddress());
user.setActiveWalletPrivateKey(encryptionService.encrypt(account.getPrivateKey()));
user.setActiveWalletPassword(passwordEncoder.encode(newWalletRequest.getActiveWalletP
assword())); user.setUpdateTimestamp(utcClock.millis());
user = userRepository.save(user);
return NewWalletResponse.builder().address(user.getActiveWalletAddress()).build();}

    @Override
  public ExportWalletResponse exportWallet(ExportWalletRequest exportWalletRequest) {
        User user =
userRepository.getByTelegramUserId(exportWalletRequest.getTelegramUserId());
if (passwordEncoder.matches(exportWalletRequest.getActiveWalletPassword(),
user.getActiveWalletPassword())) { return ExportWalletResponse.builder()
.activeWalletPrivateKey(encryptionService.decrypt(user.getActiveWalletPrivateKey()))
.build(); } throw new WrongPasswordRuntimeException("Password does not matches.");}

    @Override
    public void removeWallet(ExportWalletRequest exportWalletRequest) {
User user =
userRepository.getByTelegramUserId(exportWalletRequest.getTelegramUserId());
if (passwordEncoder.matches(exportWalletRequest.getActiveWalletPassword(),
user.getActiveWalletPassword())) { user.setActiveWalletPassword(null);
user.setActiveWalletPrivateKey(null); user.setActiveWalletAddress(null);
user.setUpdateTimestamp(utcClock.millis()); userRepository.save(user); return;}
throw new WrongPasswordRuntimeException("Password does not matches.");}

    @Override
    public void updateUser(UserUpdateRequest userUpdateRequest) {
User user =
userRepository.getByTelegramUserId(userUpdateRequest.getTelegramUserId());
if (userUpdateRequest.getActiveWalletOldPassword() != null) {
if(passwordEncoder.matches(userUpdateRequest.getActiveWalletOldPassword(),
user.getActiveWalletPassword())) {
user.setActiveWalletPassword(passwordEncoder.encode(userUpdateRequest.getActiveWallet
NewPassword()));} else { throw new WrongPasswordRuntimeException("Password does not
matches."); }} user.setUpdateTimestamp(utcClock.millis()); }
```

```
private User prepareNewUserRecord(NewWalletRequest newWalletRequest) {
long currentTimestamp = utcClock.millis();
return User.builder() .telegramUserId(newWalletRequest.getTelegramUserId())
.chatId(newWalletRequest.getChatId()).createTimestamp(currentTimestamp)
.updateTimestamp(currentTimestamp).build();}}
```

Клас CryptoAES

```
package edu.diploma.ms.core.util;
import edu.diploma.ms.core.exception.BaseRuntimeException;
import edu.diploma.ms.core.exception.ExceptionCode;
import lombok.AccessLevel; import lombok.NoArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.output.ByteArrayOutputStream;
import javax.crypto.Cipher; import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec; import java.nio.ByteBuffer;
import java.nio.CharBuffer; import java.nio.charset.StandardCharsets;
import java.util.Arrays; import java.util.Base64;
import java.util.concurrent.ThreadLocalRandom;

@Slf4j @NoArgsConstructor(access = AccessLevel.PRIVATE)
public final class CryptoAES {
private static final String CRYPTO_METHODS = "AES/ECB/PKCS5Padding";
private static final int INITIALIZATION_VECTOR_LENGTH = 12;

public static char[] encryptWithRandomIv(char[] data, String key) {
        return bytesToChar(encryptWithRandomIv(charToBytes(data), key));}
public static char[] decryptWithRandomIv(char[] data, String key) {
        return bytesToChar(decryptWithRandomIv(charToBytes(data), key));}

public static byte[] encryptWithRandomIv(byte[] data, String key) {
byte[] ivBytes = new byte[INITIALIZATION_VECTOR_LENGTH];
try (ByteArrayOutputStream outputStream = new ByteArrayOutputStream()) {
ivBytes = generateIvBytes();

SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(StandardCharsets.UTF_8),
"AES"); GCMParameterSpec parameterSpec = new GCMParameterSpec(128, ivBytes);
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, parameterSpec);

byte[] encrypted = cipher.doFinal(data);
outputStream.write(ivBytes); outputStream.write(encrypted);
return Base64.getEncoder().encode(outputStream.toByteArray());
} catch (Exception e) { String message = "Error when encrypting bytes: " +
e.getMessage(); throw new BaseRuntimeException(ExceptionCode.UNKNOWN_EXCEPTION);}}

public static byte[] decryptWithRandomIv(byte[] data, String key) {
byte[] dataBytes = Base64.getDecoder().decode(data);
byte[] ivBytes = Arrays.copyOfRange(dataBytes, 0, INITIALIZATION_VECTOR_LENGTH);
byte[] encryptedData = Arrays.copyOfRange(dataBytes, INITIALIZATION_VECTOR_LENGTH,
dataBytes.length);  try {
SecretKeySpec keySpec = new SecretKeySpec(key.getBytes(StandardCharsets.UTF_8),
"AES"); GCMParameterSpec parameterSpec = new GCMParameterSpec(128, ivBytes);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.DECRYPT_MODE, keySpec, parameterSpec);
return cipher.doFinal(encryptedData); } catch (Exception e) {
String message = "Error when decrypting bytes: " + e.getMessage();
throw new BaseRuntimeException(ExceptionCode.UNKNOWN_EXCEPTION); }}
```

```
private static byte[] charToBytes(char[] chars) {
CharBuffer charBuffer = CharBuffer.wrap(chars);
ByteBuffer byteBuffer = StandardCharsets.UTF_8.encode(charBuffer);
byte[] result = new byte[byteBuffer.remaining()];
byteBuffer.get(result); return result; }

private static char[] bytesToChar(byte[] bytes) {
ByteBuffer byteBuffer = ByteBuffer.wrap(bytes);
CharBuffer charBuffer = StandardCharsets.UTF_8.decode(byteBuffer);
char[] result = new char[charBuffer.remaining()];
charBuffer.get(result); return result;}

private static byte[] generateIvBytes() { byte[] result = new
byte[INITIALIZATION_VECTOR_LENGTH]; ThreadLocalRandom.current().nextBytes(result);
return result;}}
```

Клас TransactionController

```
package edu.diploma.ms.core.web.controller;
import edu.diploma.ms.core.integration.tron.model.BandwidthPriceTronResponse;
import edu.diploma.ms.core.integration.tron.model.TransactionTronResponse;
import edu.diploma.ms.core.service.TransactionService;
import edu.diploma.ms.core.web.model.*; import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j; import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*; import java.util.List;

@Slf4j @RequiredArgsConstructor @RestController
@RequestMapping("/api/v1/transaction") public class TransactionController {
private final TransactionService transactionService;

@PostMapping("/transfer-currency")
public TransactionTronResponse transferCurrency(@RequestBody CurrencyTransferRequest
currencyTransferRequest) {
log.info("Request to transfer currency: {}", currencyTransferRequest);
TransactionTronResponse response =
transactionService.transferCurrency(currencyTransferRequest);
log.info("Request to transfer currency processed. Transaction hash: {}",
response.getHash()); return response;}

@PostMapping("/transfer-currency/calculate-price")
public BandwidthPriceTronResponse transferCurrencyCalculatePrice(@RequestBody
CurrencyTransferRequest currencyTransferRequest) {
log.info("Request to calculate price for transfer currency: {}",
currencyTransferRequest); BandwidthPriceTronResponse response =
transactionService.calculateTransferCurrency(currencyTransferRequest);
log.info("Request to calculate price for transfer currency processed. Price: {}",
response.getBandwidthPrice()); return response;}

@PostMapping("/freeze-balance")
public TransactionTronResponse freezeBalance(@RequestBody FreezeBalanceRequest
request) { log.info("Request to freeze balance: {}", request);
TransactionTronResponse response = transactionService.freezeBalance(request);
log.info("Request to freeze balance processed. Transaction hash: {}",
response.getHash()); return response;}

@PostMapping("/freeze-balance/calculate-price")
public BandwidthPriceTronResponse freezeBalanceCalculatePrice(@RequestBody
FreezeBalanceRequest request) {
log.info("Request to calculate price for freeze balance: {}", request);
```

```
BandwidthPriceTronResponse response =
transactionService.calculateFreezeBalance(request);
log.info("Request to calculate price for freeze balance processed. Price: {}",
response.getBandwidthPrice()); return response; }

@PostMapping("/unfreeze-balance")
public TransactionTronResponse unfreezeBalance(@RequestBody UnfreezeBalanceRequest
request) { log.info("Request to unfreeze balance: {}", request);
TransactionTronResponse response = transactionService.unfreezeBalance(request);
log.info("Request to unfreeze balance processed. Transaction hash: {}",
response.getHash()); return response;}

@PostMapping("/unfreeze-balance/calculate-price")
public BandwidthPriceTronResponse unfreezeBalanceCalculatePrice(@RequestBody
UnfreezeBalanceRequest request) {
log.info("Request to calculate price for unfreeze balance: {}", request);
BandwidthPriceTronResponse response =
transactionService.calculateUnfreezeBalance(request);
log.info("Request to calculate price for unfreeze balance processed. Price: {}",
response.getBandwidthPrice()); return response; }

@PostMapping("/process-deposit-transactions")
public ResponseEntity<Void> processNewTransactions(@RequestBody
List<TransactionTronResponse> newTransactions) {
log.info("New transactions to process came. Transactions number: {}",
newTransactions.size());
transactionService.processNewDepositTransactions(newTransactions);
log.info("New transactions processed."); return ResponseEntity.ok().build(); }

@GetMapping("/last-processed-block") public Block getLastProcessedBlock() {
log.info("Request to get last processed block."); Block block =
transactionService.getLastProcessedBlock(); log.info("Request to get last processed
block processed. Block: {}", block); return block;}}
```

Клас UserController

```
package edu.diploma.ms.core.web.controller;
import edu.diploma.ms.core.service.UserService;
import edu.diploma.ms.core.web.model.*; import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j; import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@Slf4j @RequiredArgsConstructor @RestController @RequestMapping("/api/v1/user")
public class UserController {
    private final UserService userService;
@GetMapping
public UserResponse findUser(@RequestParam long telegramUserId) {
log.info("Request to find user by telegram user id: {}", telegramUserId);
UserResponse userResponse = userService.findByTelegramUserId(telegramUserId);
log.info("Request to find user by telegram user id processed.");
return userResponse; }

@PostMapping("/create-new-wallet")
public NewWalletResponse createNewWallet(@RequestBody NewWalletRequest
newWalletRequest) { log.info("Request to add new wallet: {}", newWalletRequest);
NewWalletResponse newWalletResponse = userService.createWallet(newWalletRequest);
log.info("Request to add new wallet processed. Address: {}",
newWalletResponse.getAddress()); return newWalletResponse; }

@PostMapping("/export-wallet")
```

```
public ExportWalletResponse exportWallet(@RequestBody ExportWalletRequest
exportWalletRequest) { log.info("Request to export wallet: {}", exportWalletRequest);
ExportWalletResponse exportWalletResponse =
userService.exportWallet(exportWalletRequest);
log.info("Request to export wallet processed."); return exportWalletResponse;}

@PostMapping("/remove-wallet")
public ResponseEntity<Void> removeWallet(@RequestBody ExportWalletRequest
exportWalletRequest) { log.info("Request to remove wallet: {}", exportWalletRequest);
userService.removeWallet(exportWalletRequest);
log.info("Request to remove wallet processed."); return ResponseEntity.ok().build();}

@PutMapping("/update-user")
public ResponseEntity<Void> updateUser(@RequestBody UserUpdateRequest
userUpdateRequest) { log.info("Request to update user: {}", userUpdateRequest);
userService.updateUser(userUpdateRequest);
log.info("Request to update user processed."); return ResponseEntity.ok().build();}}
```

Клас Block

```
package edu.diploma.ms.core.web.model; import lombok.*;
@Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor
public class Block { private long lastProcessedBlock;}
```

Клас CurrencyTransferRequest

```
package edu.diploma.ms.core.web.model;
import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class CurrencyTransferRequest { private long telegramUserId; private String
to; private double amount; private String memo; }
```

Клас ExportWalletRequest

```
package edu.diploma.ms.core.web.model;
import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class ExportWalletRequest { private long telegramUserId;
@ToString.Exclude private String activeWalletPassword;}
```

Клас ExportWalletResponse

```
package edu.diploma.ms.core.web.model;
import lombok.*;
@Getter @Setter @Builder
@NoArgsConstructor @AllArgsConstructor

public class ExportWalletResponse {private String activeWalletPrivateKey;}
```

Клас FreezeBalanceRequest

```
package edu.diploma.ms.core.web.model; import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class FreezeBalanceRequest {private long telegramUserId; private int amount;}
```

Клас NewWalletRequest

```
package edu.diploma.ms.core.web.model; import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class NewWalletRequest { private long telegramUserId; private long chatId;
@ToString.Exclude
private String activeWalletPrivateKey; private String activeWalletPassword;}
```

Клас NewWalletResponse

```
package edu.diploma.ms.core.web.model; import lombok.*;
@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
public class NewWalletResponse {private String address;}package
```

Клас UnfreezeBalanceRequest

```
edu.diploma.ms.core.web.model; import lombok.*;
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class UnfreezeBalanceRequest { private long telegramUserId;}
```

Клас UserResponse

```
package edu.diploma.ms.core.web.model;
import edu.diploma.ms.core.db.model.User;
import lombok.*;
@Getter @Setter @Builder @NoArgsConstructor @AllArgsConstructor
public class UserResponse {
private long telegramUserId; private long chatId; private String languageCode;
private String activeWalletAddress; private String activeWalletPassword;
public static UserResponse createOf(User user) {
return UserResponse.builder().telegramUserId(user.getTelegramUserId())
.chatId(user.getChatId()) .activeWalletAddress(user.getActiveWalletAddress())
.activeWalletPassword(user.getActiveWalletPassword()).build();}}
```

Клас UserUpdateRequest

```
package edu.diploma.ms.core.web.model;
import lombok.*;
```

```
@Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor
public class UserUpdateRequest {
private long telegramUserId; private String languageCode;
    @ToString.Exclude private String activeWalletOldPassword; //optional (with next
field) @ToString.Exclude private String activeWalletNewPassword;}
```

Клас CoreApplication

```
package edu.diploma.ms.core;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication public class CoreApplication {
public static void main(String[] args) {SpringApplication.run(CoreApplication.class,
args); }}
```

Файл властивостей application.properties

```
server.port=8666
private-key-encryption-secret-key=IVc2tsdOL2jZNU21Q9jWCHbK9vUcqweN

ms.api.path=/api
ms.tron.host=localhost
ms.tron.uri=http://${ms.tron.host}:8555

ms.tron.path.get-account-from-private-key=${ms.api.path}/v1/account/from-private
ms.tron.path.generate-account=${ms.api.path}/v1/account/generate
ms.tron.path.transfer-currency=${ms.api.path}/v1/transaction/transferCurrency
ms.tron.path.freeze-balance=${ms.api.path}/v1/transaction/freezeBalance
ms.tron.path.unfreeze-balance=${ms.api.path}/v1/transaction/unfreezeBalance
ms.tron.path.transfer-currency-calculate-bandwidth-
price=${ms.api.path}/v1/transaction/transferCurrency/calculateBandwidthPrice
ms.tron.path.freeze-balance-calculate-bandwidth-
price=${ms.api.path}/v1/transaction/freezeBalance/calculateBandwidthPrice
ms.tron.path.unfreeze-balance-calculate-bandwidth-
price=${ms.api.path}/v1/transaction/unfreezeBalance/calculateBandwidthPrice

ms.tron.timeout.connect=15000
ms.tron.timeout.read=15000
ms.bot.host=localhost
ms.bot.uri=http://${ms.bot.host}:8777
ms.bot.path.new-deposit-notification=${ms.api.path}/v1/new-deposit-transaction
ms.bot.timeout.connect=15000
ms.bot.timeout.read=15000

# Database
spring.data.mongodb.auto-index-creation=true
mongodb.db=diploma_database
mongodb.host=localhost
mongodb.port=27017
mongodb.user-name=diploma_user
mongodb.user-password=diploma_password
mongodb.connectionTimeoutMS=15000
mongodb.credentials=${mongodb.user-name}:${mongodb.user-password}@
spring.data.mongodb.uri=mongodb://${mongodb.credentials}${mongodb.host}:${mongodb.por
t}/${mongodb.db}?connectTimeoutMS=${mongodb.connectionTimeoutMS}
```

Файл збірки проекту build.gradle

```
plugins {
      id 'org.springframework.boot' version '2.4.4'
      id 'io.spring.dependency-management' version '1.0.11.RELEASE'
      id 'java'
}

group = 'edu.diploma'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'
configurations {compileOnly {extendsFrom annotationProcessor}}
repositories {mavenCentral()}

dependencies {
implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.security:spring-security-core'
implementation 'commons-io:commons-io:2.6'
compileOnly 'org.projectlombok:lombok'
annotationProcessor 'org.projectlombok:lombok'
testImplementation 'org.springframework.boot:spring-boot-starter-test'
testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo:2.2.0'}
test {useJUnitPlatform()}
```

В.2 Програмний код мікросервіса be-tron

Клас TronWrapperConfiguration

```
package edu.diploma.ms.tron.configuration;
import edu.diploma.ms.tron.properties.TronWrapperProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.tron.trident.core.ApiWrapper;
@Configuration
public class TronWrapperConfiguration {
@Bean public ApiWrapper apiWrapper(TronWrapperProperties tronWrapperProperties) {
//Official public node connection is used
return new ApiWrapper("3.218.137.187:50051", "3.218.137.187:50061",
tronWrapperProperties.getPrivateKey());}}
```

Клас TronContractType

```
package edu.diploma.ms.tron.constants;

import lombok.Getter;
import lombok.RequiredArgsConstructor;
import java.util.Arrays;
@RequiredArgsConstructor public enum TronContractType {
TRANSFER_CONTRACT(1), TRANSFER_ASSET_CONTRACT(2),
FREEZE_BALANCE_CONTRACT(11), UNFREEZE_BALANCE_CONTRACT(12),
TRIGGER_SMART_CONTRACT(31), UNKNOWN(-1), TRANSACTION_NOT_EXISTS(-1);
```

```
@Getter private final int code;
public static TronContractType findContractByCode(Integer code) {
if(code == null) {return TRANSACTION_NOT_EXISTS;}
return Arrays.stream(values()).filter(type -> type.code ==
code).findAny().orElse(UNKNOWN);}}
```

Клас TronCurrencyConverter

```
package edu.diploma.ms.tron.converter;
import lombok.AccessLevel;
import lombok.NoArgsConstructor;
import java.math.BigDecimal;
import java.math.RoundingMode;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public final class TronCurrencyConverter {
private static final BigDecimal TRON_ATOMIC_UNITS_MULTIPLIER =
BigDecimal.valueOf(1_000_000);
public static double convertAtomicUnits(long atomicUnits) {
return atomicUnits == 0 ? 0.0 :
BigDecimal.valueOf(atomicUnits).divide(TRON_ATOMIC_UNITS_MULTIPLIER, 10,
RoundingMode.HALF_DOWN).doubleValue();}
public static long convertToAtomicUnits(double currencyAmount) {
return currencyAmount == 0 ? 0 :
BigDecimal.valueOf(currencyAmount).multiply(TRON_ATOMIC_UNITS_MULTIPLIER).longValue()
;}}
```

Клас AccountDataTronScanResponse

```
package edu.diploma.ms.tron.integration.tronscan.model;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties; import lombok.*;
@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public class AccountDataTronScanResponse {
private long balance; private long totalFrozen; private Bandwidth bandwidth;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class Bandwidth {
private double freeNetLimit; private double freeNetRemaining;
private double netLimit; private double netRemaining; private double freeNetUsed;
private long totalNetWeight; private long totalNetLimit;}}
```

Клас BlockDataTronScanResponse

```
package edu.diploma.ms.tron.integration.tronscan.model;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.*;

@Getter @Setter @ToString
@NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown = true) public class
BlockDataTronScanResponse
{ private long number;}
```

Клас TransactionDataTronScanResponse

```
package edu.diploma.ms.tron.integration.tronscan.model;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.*; import java.util.List;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public class TransactionDataTronScanResponse {
private String hash; private long block; private long timestamp; private Integer
contractType; private boolean confirmed; private String contractRet;
private Cost cost; private List<Trc20TransferInfo> trc20TransferInfo;
private ContractData contractData; private String data; private Info info;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class Cost {
@JsonProperty("net_fee") private Long netFee; // burned trx for bandwidth
@JsonProperty("energy_fee") private Long energyFee; // burned trx for energy
@JsonProperty("energy_usage_total") private Long energyUsageTotal; // energy used
@JsonProperty("net_usage") private Long netUsage; // bandwidth used}

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class Trc20TransferInfo { private String symbol; private String
name; @JsonProperty("amount_str") private Long amount; @JsonProperty("to_address")
private String toAddress; @JsonProperty("from_address") private String fromAddress;}

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class ContractData { private Long amount;
@JsonProperty("to_address") private String toAddress;
@JsonProperty("owner_address") private String ownerAddress;
@JsonProperty("contract_address") private String contractAddress;
private TokenInfo tokenInfo; @JsonProperty("frozen_balance") private long
frozenBalance;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class TokenInfo { private String tokenAbbr; private String
tokenName;}}

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class Info { @JsonProperty("unfreeze_amount") private long
unfreezeAmount;}}
```

Клас TranasctionHistoryTronScanResponse

```
package edu.diploma.ms.tron.integration.tronscan.model;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.*;
import java.util.List;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public class TransactionHistoryTronScanResponse {
private Long total; private List<TransactionData> data;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class TransactionData {
private String hash; private long block; private long timestamp; private int
contractType; private boolean confirmed; private String contractRet; private String
```

```
ownerAddress; private String toAddress; private Long amount; private Cost cost;
private TokenInfo tokenInfo;

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class Cost {
@JsonProperty("net_fee") private Long netFee = 0L; // burned trx for bandwidth
@JsonProperty("energy_fee") private Long energyFee = 0L; // burned trx for energy
@JsonProperty("energy_usage_total") private Long energyUsageTotal = 0L; // energy
used @JsonProperty("net_usage") private Long netUsage = 0L; // bandwidth used}

@Getter @Setter @ToString @NoArgsConstructor @JsonIgnoreProperties(ignoreUnknown =
true) public static class TokenInfo { private String tokenAbbr; private String
tokenName;}}}
```

Клас TronWrapperProperties

```
package edu.diploma.ms.tron.properties;
import lombok.*;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;
@Getter @Setter @Component @ConfigurationProperties(prefix = "tron.wrapper")
public class TronWrapperProperties { private String privateKey;}
```

Клас NewIncomeTransactionProcessor

```
package edu.diploma.ms.tron.service.scheduled;
import edu.diploma.ms.tron.constants.*;
import edu.diploma.ms.tron.converter.TransactionConverter;
import edu.diploma.ms.tron.integration.core.communication.CoreCommunicationService;
import
edu.diploma.ms.tron.integration.tronscan.communication.TronScanCommunicationService;
import edu.diploma.ms.tron.integration.tronscan.model.BlockDataTronScanResponse;
import
edu.diploma.ms.tron.integration.tronscan.model.TransactionHistoryTronScanResponse;
import edu.diploma.ms.tron.web.model.TransactionDto;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import java.util.*;
import java.util.stream.Collectors;

@Slf4j @RequiredArgsConstructor @Service
public class NewIncomeTransactionProcessor {
private final TronScanCommunicationService tronScanCommunicationService;
private final CoreCommunicationService coreCommunicationService;

@Scheduled(initialDelay = 1000L, fixedDelay = 5000)
public void searchNewTrxTransferTransactions() {
log.info("Searching for new trx transactions."); try { search(); }
catch (Exception e) { log.error("Error when searching new trx transactions: {}",
e.getMessage());}}


private void search() throws InterruptedException {
long lastProcessedBlock =
coreCommunicationService.getBlock().getLastProcessedBlock();
```

```
BlockDataTronScanResponse latestBlockNumber =
tronScanCommunicationService.getLatestBlockNumber();
long lastBlockchainBlockToProcess = latestBlockNumber.getNumber() - 19;
TransactionDto emptyTransactionForBlockUpdate = TransactionDto.builder()
.to("").block(latestBlockNumber.getNumber() - 19).build();
if(lastProcessedBlock == 0) {
coreCommunicationService.sendNewDepositTransactions(Collections.singletonList(emptyTr
ansactionForBlockUpdate)); return;}

List<TransactionHistoryTronScanResponse.TransactionData> transactionDataList = new
ArrayList<>();
while (lastBlockchainBlockToProcess > lastProcessedBlock) {
TransactionHistoryTronScanResponse transactionsByBlock =
tronScanCommunicationService.getTransactionsByBlock(50, 0,
lastBlockchainBlockToProcess);
int collectedTransactions = 50;
fillListWithTrxTransactions(transactionDataList, transactionsByBlock);
while (collectedTransactions < transactionsByBlock.getTotal()) {
transactionsByBlock = tronScanCommunicationService
.getTransactionsByBlock(50, collectedTransactions, lastBlockchainBlockToProcess);
fillListWithTrxTransactions(transactionDataList, transactionsByBlock);
collectedTransactions = collectedTransactions + 50;}
lastBlockchainBlockToProcess--;
TimeUnit.MILLISECONDS.sleep(500);}

List<TransactionDto> depositTransactions = transactionDataList.stream()
.map(TransactionConverter::convertTronTransaction).collect(Collectors.toList());

if(depositTransactions.isEmpty()) {
depositTransactions.add(emptyTransactionForBlockUpdate);}

log.info("New trx transactions amount: {}", transactionDataList.size());
System.out.println(depositTransactions.get(0).getBlock());
coreCommunicationService.sendNewDepositTransactions(depositTransactions);}

private void
fillListWithTrxTransactions(List<TransactionHistoryTronScanResponse.TransactionData>
transactionDataList, TransactionHistoryTronScanResponse transactionsByBlock) {
transactionsByBlock.getData().stream()
.filter(transactionData -> transactionData.getContractType() ==
TronContractType.TRANSFER_CONTRACT.getCode())
.filter(transactionData ->
TransactionStatus.findByValue(transactionData.getContractRet()) ==
TransactionStatus.SUCCESS).forEach(transactionDataList::add);}}
```

Клас AccountServiceImpl

```
package edu.diploma.ms.tron.service;

import edu.diploma.ms.tron.exception.WrongPrivateKeyRuntimeException;
import
edu.diploma.ms.tron.integration.tronscan.communication.TronScanCommunicationService;
import edu.diploma.ms.tron.integration.tronscan.model.AccountDataTronScanResponse;
import edu.diploma.ms.tron.util.Calculator;
import edu.diploma.ms.tron.web.model.TronAccountDto;
import edu.diploma.ms.tron.web.model.TronAccountResourcesDto;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.bouncycastle.jcajce.provider.digest.Keccak;
import org.bouncycastle.util.encoders.Hex;
```

```
import org.springframework.stereotype.Service;
import org.tron.trident.core.ApiWrapper;
import org.tron.trident.crypto.SECP256K1;
import org.tron.trident.crypto.tuwenitypes.Bytes32;
import org.tron.trident.utils.Base58Check;
import java.util.List;

@Slf4j @RequiredArgsConstructor @Service
public class AccountServiceImpl implements AccountService {
private final TronScanCommunicationService tronScanCommunicationService;

@Override public TronAccountDto generateNewAccount() {
List<String> list = ApiWrapper.generateAddress();
String privateKey = list.get(1);
String publicKeyBase58 =
Base58Check.bytesToBase58(ApiWrapper.parseHex(list.get(0)).toByteArray());
return new TronAccountDto(privateKey, publicKeyBase58); }

@Override public TronAccountDto getAccountFromPrivateKey(String privateKeyHex) {
try { SECP256K1.KeyPair keyPair =
SECP256K1.KeyPair.create(SECP256K1.PrivateKey.create(Bytes32.fromHexString(privateKey
Hex)));
SECP256K1.PublicKey pubKey = keyPair.getPublicKey();
Keccak.Digest256 digest = new Keccak.Digest256();
digest.update(pubKey.getEncoded(), 0, 64);
byte[] raw = digest.digest(); byte[] rawAddr = new byte[21]; rawAddr[0] = 65;
System.arraycopy(raw, 12, rawAddr, 1, 20);
String rawAddrHex = Hex.toHexString(rawAddr);
String addressBase58 =
Base58Check.bytesToBase58(ApiWrapper.parseHex(rawAddrHex).toByteArray());
AccountDataTronScanResponse accountData =
tronScanCommunicationService.getAccountData(addressBase58);
if(accountData.getBandwidth() == null) {
throw new WrongPrivateKeyRuntimeException("account not exists"); }
return new TronAccountDto(privateKeyHex, addressBase58); } catch (Exception e) {
throw new WrongPrivateKeyRuntimeException(e.getMessage());}}

@Override
public TronAccountResourcesDto getAccountResources(String accountAddress) {
AccountDataTronScanResponse accountData =
tronScanCommunicationService.getAccountData(accountAddress);
if(accountData.getBandwidth() == null) {
return TronAccountResourcesDto.builder().exists(false).build();
} else if (accountData.getBandwidth().getTotalNetWeight() == 0) {
return TronAccountResourcesDto.builder().exists(true).build(); }
double bandwidthRate = Calculator.divide(1.0,
accountData.getBandwidth().getTotalNetWeight());
bandwidthRate = Calculator.multiply(bandwidthRate,
accountData.getBandwidth().getTotalNetLimit()); return
TronAccountResourcesDto.createOf(accountAddress, accountData, bandwidthRate);}}
```

Клас TransactionServiceImpl

```
package edu.diploma.ms.tron.service;

import com.google.protobuf.ByteString;
import edu.diploma.ms.tron.constants.TronContractType;
import edu.diploma.ms.tron.converter.TransactionConverter;
import edu.diploma.ms.tron.converter.TronCurrencyConverter;
import edu.diploma.ms.tron.exception.FailedTransactionRuntimeException;
```

```
import
edu.diploma.ms.tron.integration.tronscan.communication.TronScanCommunicationService;
import edu.diploma.ms.tron.integration.tronscan.model.*;
import edu.diploma.ms.tron.web.model.*; import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.tron.trident.core.ApiWrapper;
import org.tron.trident.core.exceptions.IllegalException;
import org.tron.trident.core.transaction.TransactionBuilder;
import org.tron.trident.crypto.SECP256K1;
import org.tron.trident.crypto.tuwenitypes.Bytes32;
import org.tron.trident.proto.Chain;
import org.tron.trident.proto.Contract;
import org.tron.trident.proto.Response;

import java.util.List; import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock; import java.util.function.Function;
import java.util.stream.Collectors;

@RequiredArgsConstructor @Service
public class TransactionServiceImpl implements TransactionService {
private final TronScanCommunicationService tronScanCommunicationService;
private final Map<String, Lock> transactionConcurrencyLockMap = new
ConcurrentHashMap<>(); private final ApiWrapper apiWrapper;

private final Map<TronContractType, Function<TransactionDataTronScanResponse,
TransactionDto>> transactionMappingStrategies
= Map.of(TronContractType.TRANSFER_CONTRACT,
TransactionConverter::convertTronTransaction,
TronContractType.TRANSFER_ASSET_CONTRACT,
TransactionConverter::convertTrc10Transaction,
TronContractType.TRIGGER_SMART_CONTRACT,
TransactionConverter::convertTrc20Transaction,
TronContractType.FREEZE_BALANCE_CONTRACT,
TransactionConverter::convertFreezeTransaction,
TronContractType.UNFREEZE_BALANCE_CONTRACT,
TransactionConverter::convertUnfreezeTransaction,
TronContractType.TRANSACTION_NOT_EXISTS,
TransactionConverter::convertTransactionThatNotExists,
TronContractType.UNKNOWN, TransactionConverter::convertUnknownTransaction);

@Override public TransactionDto transferCurrency(CurrencyTransferRequest
currencyTransferRequest) {
Lock lock =
transactionConcurrencyLockMap.computeIfAbsent(currencyTransferRequest.getFrom(), k ->
new ReentrantLock()); lock.lock();
try { Chain.Transaction transaction =
createAndSignTransferTransaction(currencyTransferRequest);
String txHash = apiWrapper.broadcastTransaction(transaction);
return TransactionDto.builder().hash(txHash).build();
} catch (RuntimeException e) { throw new
FailedTransactionRuntimeException(e.getMessage()); } finally { lock.unlock(); }}

@Override
public TransactionDto freezeBalance(FreezeBalanceRequest freezeBalanceRequest) {
try { Chain.Transaction transaction =
createAndSignFreezeTransaction(freezeBalanceRequest);
String txHash = apiWrapper.broadcastTransaction(transaction);

return TransactionDto.builder().hash(txHash).build();}
catch (RuntimeException e) { throw new
FailedTransactionRuntimeException(e.getMessage());}}

@Override
```

```
public TransactionDto unfreezeBalance(UnfreezeBalanceRequest unfreezeBalanceRequest)
{ try { Chain.Transaction transaction =
createAndSignUnfreezeTransaction(unfreezeBalanceRequest);
String txHash = apiWrapper.broadcastTransaction(transaction);
return TransactionDto.builder().hash(txHash).build(); } catch (RuntimeException e) {
throw new FailedTransactionRuntimeException(e.getMessage()); }}

@Override
public BandwidthPriceDto
calculateTransferCurrencyBandwidthPrice(CurrencyTransferRequest
currencyTransferRequest) { Chain.Transaction transaction =
createAndSignTransferTransaction(currencyTransferRequest);
return new BandwidthPriceDto(calculateSignedTransactionSize(transaction));}

@Override
public BandwidthPriceDto calculateFreezeBalanceBandwidthPrice(FreezeBalanceRequest
freezeBalanceRequest) {
Chain.Transaction transaction = createAndSignFreezeTransaction(freezeBalanceRequest);
return new BandwidthPriceDto(calculateSignedTransactionSize(transaction));}

@Override
public BandwidthPriceDto
calculateUnfreezeBalanceBandwidthPrice(UnfreezeBalanceRequest unfreezeBalanceRequest)
{ Chain.Transaction transaction =
createAndSignUnfreezeTransaction(unfreezeBalanceRequest);
return new BandwidthPriceDto(calculateSignedTransactionSize(transaction));}

@Override
public TransactionDto getTransaction(String transactionId) {
TransactionDataTronScanResponse transaction =
tronScanCommunicationService.getTransaction(transactionId);
TronContractType tronContractType =
TronContractType.findContractByCode(transaction.getContractType());
return transactionMappingStrategies.get(tronContractType).apply(transaction);}

@Override
public Page<TransactionDto> getTransactionHistory(int page, int size, String address)
{ long skip = (long) size * page;
TransactionHistoryTronScanResponse accountTransactionHistory =
tronScanCommunicationService.getAccountTransactionHistory(size, skip, address);
List<TransactionDto> data = accountTransactionHistory.getData().stream()
.map(TransactionConverter::convertHistory).collect(Collectors.toList());
return Page.createOf(data, size, accountTransactionHistory.getTotal());}

private int calculateSignedTransactionSize(Chain.Transaction transaction) {
return transaction.getSerializedSize() + transaction.getSignature(0).size() - 1;}

private Chain.Transaction createAndSignTransferTransaction(CurrencyTransferRequest
currencyTransferRequest) {
ByteString rawFromAddress =
ApiWrapper.parseAddress(currencyTransferRequest.getFrom());
ByteString rawToAddress = ApiWrapper.parseAddress(currencyTransferRequest.getTo());
Contract.TransferContract transfer = Contract.TransferContract.newBuilder()
.setOwnerAddress(rawFromAddress).setToAddress(rawToAddress)
.setAmount(TronCurrencyConverter.convertToAtomicUnits(currencyTransferRequest.getAmou
nt())).build();

SECP256K1.KeyPair privateKeyPair =
getKeyPairFromPrivateKey(currencyTransferRequest.getPrivateKey());
Chain.Transaction transaction = apiWrapper.blockingStub.createTransaction(transfer);
TransactionBuilder transactionBuilder = new TransactionBuilder(transaction);
if (currencyTransferRequest.getMemo() != null) {
transactionBuilder.setMemo(currencyTransferRequest.getMemo());}
transaction = transactionBuilder.build();
```

```
transaction = apiWrapper.signTransaction(transaction, privateKeyPair);
return transaction;}

private Chain.Transaction createAndSignFreezeTransaction(FreezeBalanceRequest
freezeBalanceRequest) {
long atomicUnitsToFreeze =
TronCurrencyConverter.convertToAtomicUnits(freezeBalanceRequest.getAmount());
int resourceCode = freezeBalanceRequest.getResource().getCode();
String addressTo = freezeBalanceRequest.getReceiverAddress() == null ? "" :
freezeBalanceRequest.getReceiverAddress();
Response.TransactionExtention freezeTransaction;
try {
freezeTransaction = apiWrapper.freezeBalance(freezeBalanceRequest.getOwnerAddress(),
atomicUnitsToFreeze, 3, resourceCode, addressTo);
} catch (IllegalException e) {throw new FailedTransactionRuntimeException("Failed to
create freeze balance exception: " + e.getMessage()); }
SECP256K1.KeyPair privateKeyPair =
getKeyPairFromPrivateKey(freezeBalanceRequest.getPrivateKey());
return apiWrapper.signTransaction(freezeTransaction, privateKeyPair);}

private Chain.Transaction createAndSignUnfreezeTransaction(UnfreezeBalanceRequest
unfreezeBalanceRequest) { Response.TransactionExtention unfreezeTransaction;
try { unfreezeTransaction =
apiWrapper.unfreezeBalance(unfreezeBalanceRequest.getOwnerAddress(),
unfreezeBalanceRequest.getResource().getCode()); } catch (IllegalException e) { throw
new FailedTransactionRuntimeException("Failed to create unfreeze balance exception: "
+ e.getMessage());} SECP256K1.KeyPair privateKeyPair =
getKeyPairFromPrivateKey(unfreezeBalanceRequest.getPrivateKey());
return apiWrapper.signTransaction(unfreezeTransaction, privateKeyPair);}

private SECP256K1.KeyPair getKeyPairFromPrivateKey(String privateKeyHex) { return
SECP256K1.KeyPair.create(SECP256K1.PrivateKey.create(Bytes32.fromHexString(privateKey
Hex)));}}
```

Клас Calculator

```
package edu.diploma.ms.tron.util;
import lombok.AccessLevel;
import lombok.NoArgsConstructor;
import java.math.BigDecimal;
import java.math.RoundingMode;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public final class Calculator {
public static double divide(double numberToDivide, double divisor) {
BigDecimal divide =
BigDecimal.valueOf(numberToDivide).divide(BigDecimal.valueOf(divisor), 20,
RoundingMode.HALF_UP); return divide.doubleValue();}

public static double multiply(double first, double second) {
return BigDecimal.valueOf(first).multiply(BigDecimal.valueOf(second))
.setScale(5, RoundingMode.HALF_DOWN).doubleValue();}}
```

Клас AccountController

```
package edu.diploma.ms.tron.configuration;
import edu.diploma.ms.tron.properties.TronWrapperProperties;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.tron.trident.core.ApiWrapper;
@Configuration
public class TronWrapperConfiguration {
@Bean public ApiWrapper apiWrapper(TronWrapperProperties tronWrapperProperties) {
//Official public node connection is used
return new ApiWrapper("3.218.137.187:50051", "3.218.137.187:50061",
tronWrapperProperties.getPrivateKey());}}
```

Клас TransactionController

```
package edu.diploma.ms.tron.configuration;
import edu.diploma.ms.tron.properties.TronWrapperProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.tron.trident.core.ApiWrapper;
@Configuration
public class TronWrapperConfiguration {
@Bean public ApiWrapper apiWrapper(TronWrapperProperties tronWrapperProperties) {
//Official public node connection is used
return new ApiWrapper("3.218.137.187:50051", "3.218.137.187:50061",
tronWrapperProperties.getPrivateKey());}}
```

В.3 Програмний код мікросервіса be-bot

Клас SessionService

```
package edu.diploma.ms.bot.business.service;

import edu.diploma.ms.bot.business.constants.SessionState;
import edu.diploma.ms.bot.business.converter.UserSessionConverter;
import edu.diploma.ms.bot.business.db.model.UserSessionModel;
import edu.diploma.ms.bot.business.db.repository.UserSessionModelRepository;
import edu.diploma.ms.bot.business.exception.NotFoundRuntimeException;
import
edu.diploma.ms.bot.business.integration.core.communication.CoreCommunicationService;
import edu.diploma.ms.bot.business.integration.core.model.UserCoreResponse;
import edu.diploma.ms.bot.engine.entity.*;
import edu.diploma.ms.bot.engine.entity.enums.InputType;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j; import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.User;

import java.time.Clock; import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

@Slf4j @RequiredArgsConstructor @Service
public class SessionService {
private static final String START_KEYBOARD_REDIRECT_COMMAND = "/start";

private final Map<Long, UserSession> sessionMap = new ConcurrentHashMap<>();
private final Clock utcConfiguredClock;
private final UserSessionModelRepository userSessionModelRepository;
```

```
private final CoreCommunicationService coreCommunicationService;

public UserSession prepareSession(Update update, Long chatId, String data) {
UserSession userSession = sessionMap.get(chatId); if (userSession == null) {
UserSession preparedUserSession = prepareSessionOrCreate(update, chatId, data);
sessionMap.put(chatId, preparedUserSession);
return preparedUserSession;}
setInputToSessionFromUpdate(update, data, userSession);
return userSession;}

private UserSession prepareSessionOrCreate(Update update, Long chatId, String data) {
return userSessionModelRepository.findByChatId(chatId)
.map(userSessionModel -> setInputForDatabaseSession(update, data, userSessionModel))
.orElseGet(() -> createSession(update, chatId));}

private UserSession setInputForDatabaseSession(Update update, String data,
UserSessionModel model) {
UserSession userSessionFromDb = UserSessionConverter.convert(model);
setInputToSessionFromUpdate(update, data, userSessionFromDb);
return userSessionFromDb;}

private void setInputToSessionFromUpdate(Update update, String data, UserSession
userSession) { if (update.hasCallbackQuery()) { userSession.setCallbackData(data);
} else { if (data == null) { userSession.setInputType(InputType.UNSUPPORTED);
} else { userSession.setInputType(InputType.TEXT); userSession.setInputData(data);}}}

private UserSession createSession(Update update, Long chatId) {
User user; if (update.hasCallbackQuery()) {user =
update.getCallbackQuery().getFrom();} else {user = update.getMessage().getFrom();}

UserSession.UserSessionBuilder newSessionBuilder = UserSession.builder();
TelegramUserInfo profile;

profile = TelegramUserInfo.builder().userId(Long.valueOf(user.getId()))
.chatId(chatId).fullName(String.format("%s %s", user.getFirstName(),
user.getLastName())).userName(user.getUserName()).build();

UserSession newUserSession = newSessionBuilder.telegramUserInfo(profile)
.paginationState(new PaginationState()).build();

UserCoreResponse userDataFromCore = coreCommunicationService.findUser(user.getId());
if (userDataFromCore.getActiveWalletAddress() != null) {
newUserSession.setActiveWalletAddress(userDataFromCore.getActiveWalletAddress());
newUserSession.setState(SessionState.HAS_ACTIVE_WALLET); } else
{newUserSession.setState(SessionState.HAS_NO_ACTIVE_WALLET);}
newUserSession.setInputData(START_KEYBOARD_REDIRECT_COMMAND);
newUserSession.setInputType(InputType.TEXT);

Long createDate = utcConfiguredClock.millis();
userSessionModelRepository.save(UserSessionConverter.convert(newUserSession, chatId,
createDate)); log.info("Successfully created session for user with id {}.",
user.getId()); return newUserSession;}

public void update(UserSession userSession, Long chatId) {
UserSessionModel toUpdate =
userSessionModelRepository.findByChatId(chatId).orElseThrow(() ->

new NotFoundRuntimeException(String.format("Session not found. Chat id: %d",
chatId)));

UserSessionModel userSessionModel = UserSessionConverter.convert(userSession, chatId,
toUpdate.getCreatedDate(), utcConfiguredClock.millis(), toUpdate.getId());
userSessionModelRepository.save(userSessionModel);}}
```

Клас UserSession

```
package edu.diploma.ms.bot.engine.entity;
import edu.diploma.ms.bot.business.constants.SessionState;
import edu.diploma.ms.bot.business.integration.core.model.*;
import edu.diploma.ms.bot.business.integration.tron.model.*;
import edu.diploma.ms.bot.engine.entity.config.ButtonConfiguration;
import edu.diploma.ms.bot.engine.entity.config.MessageConfiguration;
import edu.diploma.ms.bot.engine.entity.enums.InputType;
import edu.diploma.ms.bot.engine.entity.pagination.PaginationState;
import lombok.*;
@Data @Builder
public class UserSession {

private TelegramUserInfo telegramUserInfo; private String callbackData; private
String inputData; private Integer lastInlineKeyboardMessageId;
private Integer lastReplyKeyboardMessageId; private Integer lastMessageId;

private PaginationState paginationState; private InputType inputType;
private MessageConfiguration messageConfiguration; private ButtonConfiguration
activeButton; private SessionState state; private String activeWalletAddress;

private AccountResourcesTronResponse selectedAccount;
private TransactionTronResponse selectedTransaction;
private NewWalletCoreRequest createWalletRequest;
private NewWalletCoreRequest importWalletRequest;
private CurrencyTransferCoreRequest currencyTransferCoreRequest;
private FreezeBalanceCoreRequest freezeBalanceCoreRequest;}
```

Клас TelegramUserInfo

```
package edu.diploma.ms.bot.engine.entity;
import lombok.Builder;
import lombok.Data;
@Data @Builder
public class TelegramUserInfo { private String fullName; private Long userId; private
String userName; private Long chatId;}
```

Клас ResponseMessageWrapper

```
package edu.diploma.ms.bot.engine.entity;
import java.util.List;
import edu.diploma.ms.bot.engine.entity.enums.MessageProcessStrategy;
import lombok.Builder; import lombok.Data;
import org.springframework.context.ApplicationContext;

@Data @Builder
public class ResponseMessageWrapper {

private ApplicationContext ctx; private UserSession userSession;
private Integer messageId; private MessageProcessStrategy messageProcessStrategy;
private String text; private List<List<Button>> callbacks;
private boolean responseToCommand;

public Long getUserId() {return userSession.getTelegramUserInfo().getUserId();}
```

```
public void setTextToMessageConfiguration(String message) {
userSession.getMessageConfiguration().getText().setText(message);
userSession.getMessageConfiguration().getText().setArgGenerator(null);}}
```

Клас BotGateway

```
package edu.diploma.ms.bot.engine;

import edu.diploma.ms.bot.business.service.SessionService;
import edu.diploma.ms.bot.engine.config.EngineCoreProperties;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.services.UserUpdateProcessor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.integration.support.locks.LockRegistry;
import org.springframework.stereotype.Component;
import org.telegram.abilitybots.api.bot.AbilityBot;
import org.telegram.abilitybots.api.objects.Reply;
import org.telegram.telegrambots.meta.api.objects.Update;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors; import java.util.function.Consumer;
import static org.telegram.abilitybots.api.objects.Flag.CALLBACK_QUERY;
import static org.telegram.abilitybots.api.objects.Flag.MESSAGE;

@Component
public class BotGateway extends AbilityBot {
private final EngineCoreProperties properties;
private final UserUpdateProcessor userUpdateProcessor;
private final SessionService sessionService;
private final ExecutorService updateProcessorExecutor;
private final LockRegistry lockRegistry;

@Lazy @Autowired
public BotGateway(EngineCoreProperties properties, LockRegistry
globalCustomerLockRegistry,
UserUpdateProcessor userUpdateProcessor, SessionService sessionService) {
super(properties.getToken(), properties.getUsername());
this.properties = properties; this.userUpdateProcessor = userUpdateProcessor;
this.sessionService = sessionService; this.updateProcessorExecutor =
Executors.newFixedThreadPool(properties.getUpdateReceiverThreadPoolSize());
this.lockRegistry = globalCustomerLockRegistry;}

public Reply replyToMessage() {
Consumer<Update> consumer = u -> updateProcessorExecutor.execute(() -> {
Long chatId = u.getMessage().getChatId();
lockRegistry.obtain(chatId).lock();
try { UserSession userSession = sessionService.prepareSession(u, chatId,
u.getMessage().getText()); userUpdateProcessor.processInputMessage(u, userSession);
sessionService.update(userSession, chatId); } finally {
lockRegistry.obtain(chatId).unlock();}}); return Reply.of(consumer, MESSAGE);}

public Reply replyToCallBack() {
Consumer<Update> consumer = u -> updateProcessorExecutor.execute(() -> {
Long chatId = u.getCallbackQuery().getMessage().getChatId();
lockRegistry.obtain(chatId).lock(); try { UserSession userSession =
sessionService.prepareSession(u, chatId, u.getCallbackQuery().getData());

userUpdateProcessor.processCallback(u, userSession);
sessionService.update(userSession, chatId); } finally {
lockRegistry.obtain(chatId).unlock();}}); return Reply.of(consumer, CALLBACK_QUERY);}
```

```
@Override public int creatorId() { EngineCoreProperties.Creator creator =
properties.getCreator();return creator.getId();}}
```

Клас TelegramBotApplication

```
package edu.diploma.ms.bot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;
import org.telegram.telegrambots.ApiContextInitializer;

@EnableMongoRepositories @SpringBootApplication
public class TelegramBotApplication { public static void main(String[] args) {
ApiContextInitializer.init();
SpringApplication.run(TelegramBotApplication.class, args);}}
```

Клас EngineCoreProperties

```
package edu.diploma.ms.bot.engine.config;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;
import java.util.Locale;

@Getter @Setter @ConfigurationProperties("bot") @Component
public class EngineCoreProperties {
private Integer updateReceiverLockRegistrySize;
private String token; private String username;
private Locale defaultLocale; private Integer updateReceiverThreadPoolSize;
private String messageConfigPath; private Creator creator;
@Setter @Getter @NoArgsConstructor
public static class Creator {private Integer id;}}
```

Клас UserUpdateProcessor

```
package edu.diploma.ms.bot.engine.services;

import edu.diploma.ms.bot.business.action.BotCommandActions;
import edu.diploma.ms.bot.business.action.StartActions;
import edu.diploma.ms.bot.business.constants.engine.BotCommand;
import edu.diploma.ms.bot.engine.constants.Keyboard;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.entity.config.ButtonConfiguration;
import edu.diploma.ms.bot.engine.entity.config.MessageConfigurationStorage;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Component;
```

```
import org.telegram.telegrambots.meta.api.objects.Update;
import java.util.Objects;

@Slf4j @RequiredArgsConstructor @Component
public class UserUpdateProcessor {
private static final String START_COMMAND = "/start";
private final MessageTextProcessor messageTextProcessor;
private final MessageKeyboardProcessor messageKeyboardProcessor;
private final UserActionProcessor userActionProcessor;
private final MessageSender messageSender;
private final ApplicationContext applicationContext;

public void processCallback(Update update, UserSession userSession) {
if (clickShouldNotBeProcessed(userSession)) {return;}
Integer lastMessageId = userSession.getLastMessageId();
if (Objects.isNull(lastMessageId)) { sendKeyboardIsOldMessage(update, userSession);
userSession.setInputData(START_COMMAND); processInputMessage(update, userSession);
return;}

boolean keyboardIsOld =
lastMessageId.compareTo(update.getCallbackQuery().getMessage().getMessageId()) > 0;
if (keyboardIsOld) { sendKeyboardIsOldMessage(update, userSession); return;}

setActiveButton(userSession);Keyboard next = userSession.getActiveButton().getNext();
if (next != null) {
MessageConfigurationStorage.setKeyboardToSession(userSession, next);}
ResponseMessageWrapper wrapper = prepareSessionWrapper(userSession, update);
userActionProcessor.processActionOnCallback(wrapper);
prepareMessageFromConfigurationAndSend(wrapper);}

private void sendKeyboardIsOldMessage(Update update, UserSession userSession) {
messageSender.sendEditMessageWithSuccessReturned(userSession.getTelegramUserInfo().ge
tUserId(), update.getCallbackQuery().getMessage().getMessageId(),
"Термін дії цієї клавіатури закінчився. Будь ласка, використовуйте новішу
клавіатуру.");}

public void processInputMessage(Update update, UserSession userSession) {

boolean responseToCommand = false;
String inputData = userSession.getInputData();
if (inputData != null && inputData.startsWith(START_COMMAND)) {
StartActions.startCommandAction(userSession);
} else if (BotCommand.isAllowedCommand(inputData, userSession.getState())) {
StartActions.clearInputTriggers(userSession);
BotCommand command = BotCommand.getCommandKeyboard(inputData);
Keyboard keyboard = command.getKeyboard();
MessageConfigurationStorage.setKeyboardToSession(userSession, keyboard);
if(command.isHasAction()) { BotCommandActions.processCommand(userSession, command);
responseToCommand = true;}}
if (Objects.nonNull(userSession.getLastInlineKeyboardMessageId())) {
messageSender.sendDeleteMessageWithSuccessReturned(userSession.getTelegramUserInfo().
getUserId(), userSession.getLastInlineKeyboardMessageId());
userSession.setLastInlineKeyboardMessageId(null);}

ResponseMessageWrapper wrapper = prepareSessionWrapper(userSession, update);
userActionProcessor.processActionOnInputMessage(wrapper, responseToCommand);
prepareMessageFromConfigurationAndSend(wrapper);}

private boolean clickShouldNotBeProcessed(UserSession userSession) {
return !(Objects.isNull(userSession.getActiveButton()) ||
!userSession.getCallbackData().equals(userSession.getActiveButton().getId())
|| Boolean.TRUE.equals(userSession.getActiveButton().getMultiClickable()));}

private void setActiveButton(UserSession userSession) {
```

```
ButtonConfiguration button =
userSession.getMessageConfiguration().getButtonById(userSession.getCallbackData());
if(button == null) { button = ButtonConfiguration.builder().next(Keyboard.START
).build(); }userSession.setActiveButton(button);}

private void prepareMessageFromConfigurationAndSend(ResponseMessageWrapper m) {
try { messageKeyboardProcessor.prepareKeyboard(m);
messageTextProcessor.prepareText(m); } catch (RuntimeException e) {
log.error("Error transforming message configuration to user {}: {}", m.getUserId(),
e.getMessage(), e);
MessageConfigurationStorage.setKeyboardToSession(m.getUserSession(), Keyboard.ERROR);
messageKeyboardProcessor.prepareKeyboard(m);
messageTextProcessor.prepareText(m);} messageSender.sendMessage(m);}

private ResponseMessageWrapper prepareSessionWrapper(UserSession s, Update u) {
ResponseMessageWrapper.ResponseMessageWrapperBuilder builder =
ResponseMessageWrapper.builder().userSession(s).ctx(applicationContext);
if (u.hasCallbackQuery()) {
builder.messageId(u.getCallbackQuery().getMessage().getMessageId()).text(u.getCallbac
kQuery().getMessage().getText()); } else {
builder.messageId(u.getMessage().getMessageId()).text(u.getMessage().getText());}
ResponseMessageWrapper message = builder.build();
message.getUserSession().setLastMessageId(message.getMessageId()); return message;}}
```

Клас UserActionProcessor

```
package edu.diploma.ms.bot.engine.services;

import edu.diploma.ms.bot.engine.constants.Keyboard;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.config.MessageConfigurationStorage;
import edu.diploma.ms.bot.engine.entity.enums.MessageProcessStrategy;
import edu.diploma.ms.bot.engine.exception.ValidationErrorException;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import java.util.Objects;
import java.util.Optional;
import static edu.diploma.ms.bot.engine.entity.enums.MessageProcessStrategy.*;

@Slf4j @RequiredArgsConstructor @Service
public class UserActionProcessor {

public void processActionOnInputMessage(ResponseMessageWrapper wrapper, boolean
responseToCommand) {
processAction(wrapper, CREATE_NEW, responseToCommand);}

public void processActionOnCallback(ResponseMessageWrapper wrapper) {
processAction(wrapper, EDIT_PRESENT, false);}

private void processAction(ResponseMessageWrapper wrapper, MessageProcessStrategy
messageProcessStrategy, boolean responseToCommand) {
wrapper.setMessageProcessStrategy(messageProcessStrategy);
processMethod(wrapper, responseToCommand);}

private void processMethod(ResponseMessageWrapper wrapper, boolean responseToCommand)
{Optional.ofNullable(wrapper.getUserSession().getActiveButton()).ifPresent(b -> { try
{ if (wrapper.getMessageProcessStrategy().equals(CREATE_NEW) && !responseToCommand) {
Optional.ofNullable(b.getInputValidator()).ifPresent(v -> {
v.getValidator().accept(wrapper);
```

```
Optional.ofNullable(b.getAction()).ifPresent(a -> a.getAction().accept(wrapper));});
} else if (Objects.isNull(b.getInputValidator())) {
Optional.ofNullable(b.getAction()).ifPresent(a -> a.getAction().accept(wrapper));}
} catch (ValidationErrorException error) {
String errorMessage = error.getErrorMessage();
wrapper.setTextToMessageConfiguration(errorMessage);
} catch (RuntimeException e) {
log.error("Error processing action: {}", e.getMessage());
MessageConfigurationStorage.setKeyboardToSession(wrapper.getUserSession(),
Keyboard.ERROR);}}); wrapper.setText(null);
wrapper.getUserSession().setInputData(null);}}
```

Клас MessageTextProcessor

```
package edu.diploma.ms.bot.engine.services;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.entity.config.MessageTextConfiguration;
import lombok.RequiredArgsConstructor; import org.springframework.stereotype.Service;
import java.util.Objects;
import static edu.diploma.ms.bot.engine.utils.MessageFormatUtil.formatMessage;

@RequiredArgsConstructor @Service public class MessageTextProcessor {

public void prepareText(ResponseMessageWrapper message) {
UserSession userSession = message.getUserSession();
MessageTextConfiguration messageTextConfig =
userSession.getMessageConfiguration().getText(); String text;

if (Objects.nonNull(messageTextConfig.getText())) { text =
messageTextConfig.getText(); } else if
(Objects.isNull(messageTextConfig.getMessage())) {
text = messageTextConfig.getTextGenerator().getGenerator().apply(message); } else {
Object[] textArgs = null;
if (Objects.nonNull(messageTextConfig.getArgGenerator())) { textArgs =
messageTextConfig.getArgGenerator().getGenerator().apply(message.getUserSession());}
text = formatMessage(messageTextConfig.getMessage(), textArgs);
} message.setText(text);}}
```

Клас MessageKeyboardProcessor

```
package edu.diploma.ms.bot.engine.services;

import edu.diploma.ms.bot.business.constants.engine.KeyboardType;
import edu.diploma.ms.bot.engine.entity.Button;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.entity.config.ButtonConfiguration;
import edu.diploma.ms.bot.engine.entity.config.ButtonLabelConfiguration;
import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;
```

```
import static edu.diploma.ms.bot.engine.utils.MessageFormatUtil.formatMessage;

@Slf4j @RequiredArgsConstructor @Service
public class MessageKeyboardProcessor {

private final MessageKeyboardPaginationProcessor messageKeyboardPaginationProcessor;

public void prepareKeyboard(ResponseMessageWrapper message) {
if(message.getUserSession().getMessageConfiguration().getKeyboardType() ==
KeyboardType.INLINE) {prepareInlineKeyboard(message);
} else { prepareReplyKeyboard(message);}}

private void prepareReplyKeyboard(ResponseMessageWrapper message) {
UserSession userSession = message.getUserSession();
List<List<ButtonConfiguration>> callbacks = userSession
.getMessageConfiguration().getCallbacks();
message.setCallbacks(callbacks.stream()
.map(row -> row.stream().map(buttonConf -> Button.builder()
.key(prepareTextFromConfiguration(buttonConf, message))
.id(buttonConf.getId()).build()).collect(Collectors.toList())).filter(row ->
!row.isEmpty()).collect(Collectors.toList()));}

private void prepareInlineKeyboard(ResponseMessageWrapper message) {
UserSession userSession = message.getUserSession();

messageKeyboardPaginationProcessor.preparePageKeyboardIfPresent(message);
List<List<ButtonConfiguration>> callbacks =
userSession.getMessageConfiguration().getCallbacks();

message.setCallbacks(callbacks.stream().map(row -> row.stream()
.filter(buttonConf -> filterButtonDynamicCondition(buttonConf,
message)).map(buttonConfiguration -> Button.builder()
.key(prepareTextFromConfiguration(buttonConfiguration, message))
.id(buttonConfiguration.getId()).build()).collect(Collectors.toList()))
.filter(row -> !row.isEmpty()).collect(Collectors.toList()));}

private String prepareTextFromConfiguration(ButtonConfiguration b,
ResponseMessageWrapper message) {
ButtonLabelConfiguration label = b.getLabel();
if (Objects.nonNull(label.getText())) { return label.getText();
} else { if (Objects.nonNull(label.getTextGenerator())) {
return label.getTextGenerator().getGenerator().apply(message, b);
} else { return formatMessage(label.getMessage());}}}

private boolean filterButtonDynamicCondition(ButtonConfiguration b,
ResponseMessageWrapper message) {
if (Objects.isNull(b.getDynamicValidator())) { return true;}
return b.getDynamicValidator().getValidator().test(message, b);}}
```

Клас MessageKeyboardPaginationProcessor

```
package edu.diploma.ms.bot.engine.services;
import edu.diploma.ms.bot.engine.constants.Action;
import edu.diploma.ms.bot.engine.constants.ButtonDynamicValidator;
import edu.diploma.ms.bot.engine.constants.Keyboard;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.config.ButtonConfiguration;
import edu.diploma.ms.bot.engine.entity.config.ButtonLabelConfiguration;
import edu.diploma.ms.bot.engine.entity.config.MessageConfigurationStorage;
import edu.diploma.ms.bot.engine.entity.enums.TypeOfButton;
```

```
import edu.diploma.ms.bot.engine.entity.pagination.EntityForPagination;
import edu.diploma.ms.bot.engine.entity.pagination.PaginationPage;
import edu.diploma.ms.bot.engine.entity.pagination.PaginationPreparationService;
import edu.diploma.ms.bot.engine.entity.pagination.PaginationState;
import lombok.RequiredArgsConstructor; import lombok.extern.slf4j.Slf4j;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;
import javax.annotation.Resource; import java.util.*;

@Slf4j @RequiredArgsConstructor @Service
public class MessageKeyboardPaginationProcessor {

private static final Integer PAGE_SIZE_DEFAULT = 5;
private static final ButtonConfiguration NEXT_BUTTON = ButtonConfiguration.builder()
.id("PAGINATION_NEXT_PAGE")
.label(ButtonLabelConfiguration.builder().text("> далі").build())
.dynamicValidator(ButtonDynamicValidator.NEXT_PAGE_VALIDATOR)
.action(Action.NEXT_PAGE_ACTION).multiClickable(true).build();
private static final ButtonConfiguration PREVIOUS_BUTTON =
ButtonConfiguration.builder().id("PAGINATION_PREVIOUS_PAGE")
.label(ButtonLabelConfiguration.builder().text("назазд <").build())
.dynamicValidator(ButtonDynamicValidator.PREVIOUS_PAGE_VALIDATOR)
.action(Action.PREVIOUS_PAGE_ACTION).multiClickable(true).build();

@Resource private ApplicationContext ctx;

public void preparePageKeyboardIfPresent(ResponseMessageWrapper message) {
List<List<ButtonConfiguration>> buttonsConfiguration =
message.getUserSession().getMessageConfiguration().getCallbacks();

buttonsConfiguration.stream()
.filter(row -> TypeOfButton.PAGINATION == row.get(0).getType()).findFirst()
.ifPresent(paginationConfigRow -> prepareButtons(message, buttonsConfiguration,
paginationConfigRow.get(0), buttonsConfiguration.indexOf(paginationConfigRow)));}

private void prepareButtons(ResponseMessageWrapper message,
List<List<ButtonConfiguration>> buttonsConfiguration,
ButtonConfiguration paginationConfig, Integer paginationConfigRowIndex) {
PaginationState paginationState = message.getUserSession().getPaginationState();
fillPaginationStateWithData(message);
List<EntityForPagination> paginationValues = paginationState.getPaginationValues();

List<List<ButtonConfiguration>> paginationConfigReplacement = new ArrayList<>();

paginationValues.stream().filter(EntityForPagination::displayAsButton)
.forEach(paginationValue ->
paginationConfigReplacement.add(Collections.singletonList(ButtonConfiguration.builder
().next(paginationConfig.getNext()).inputValidator(paginationConfig.getInputValidator
()).action(paginationConfig.getAction()).id(String.format("PAGE_ELEMENT_%d",
paginationValues.indexOf(paginationValue)))

.label(ButtonLabelConfiguration.createOf(getTextForPaginationButton(paginationConfig,
message, paginationValue))).build())));
paginationConfigReplacement.add(createPreviousAndNextPageButtons(message.getUserSessi
on().getActiveButton().getNext()));

buttonsConfiguration = new ArrayList<>(buttonsConfiguration);
buttonsConfiguration.remove(paginationConfigRowIndex.intValue());
buttonsConfiguration.addAll(paginationConfigRowIndex, paginationConfigReplacement);
message.getUserSession().getMessageConfiguration().setCallbacks(buttonsConfiguration)
;}

private void fillPaginationStateWithData(ResponseMessageWrapper message) {
PaginationState paginationState = message.getUserSession().getPaginationState();
```

```
PaginationPreparationService paginationPreparationService =
(PaginationPreparationService) ctx
.getBean(paginationState.getPaginationServiceName());
if (Objects.isNull(paginationState.getPageSize())) {
paginationState.setPageSize(PAGE_SIZE_DEFAULT);}
PaginationPage paginationPage =paginationPreparationService.getByPagination(message);
paginationState.setPaginationValues(paginationPage.getContent());
paginationState.setTotalPages(paginationPage.getTotalPages());}

private List<ButtonConfiguration> createPreviousAndNextPageButtons(Keyboard
nextKeyboardPath) {
List<ButtonConfiguration> pageSelectionRow = new ArrayList<>();
ButtonConfiguration next = MessageConfigurationStorage.copy(NEXT_BUTTON);
next.setNext(nextKeyboardPath);
ButtonConfiguration previous = MessageConfigurationStorage.copy(PREVIOUS_BUTTON);
previous.setNext(nextKeyboardPath);

pageSelectionRow.add(previous);
pageSelectionRow.add(next);
return pageSelectionRow;
}

private String getTextForPaginationButton(ButtonConfiguration configuration,
ResponseMessageWrapper message, EntityForPagination paginationValue) { try { return
configuration.getLabel().getPaginationTextGenerator().getGenerator().apply(message,
paginationValue); } catch (Exception e) {return "Помилка системи.";}}}
```

Клас MessageConfigurationStorage

```
package edu.diploma.ms.bot.engine.entity.config;

import edu.diploma.ms.bot.business.constants.engine.KeyboardType;
import edu.diploma.ms.bot.engine.constants.ButtonTextGenerator;
import edu.diploma.ms.bot.engine.constants.Keyboard;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.entity.enums.TypeOfButton;
import lombok.AccessLevel;
import lombok.NoArgsConstructor;

import java.util.List;
import java.util.function.Supplier;

import static
edu.diploma.ms.bot.engine.constants.Action.CALCULATE_FREEZE_BALANCE_ACTION;
import static
edu.diploma.ms.bot.engine.constants.Action.CALCULATE_UNFREEZE_BALANCE_ACTION;

import static edu.diploma.ms.bot.engine.constants.Action.*;
import static edu.diploma.ms.bot.engine.constants.ButtonDynamicValidator.*;
import static edu.diploma.ms.bot.engine.constants.ButtonPaginationTextGenerator.*;

import static edu.diploma.ms.bot.engine.constants.InputValidator.*;
import static edu.diploma.ms.bot.engine.constants.TextArgGenerator.*;
import static edu.diploma.ms.bot.engine.constants.TextGenerator.*;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public final class MessageConfigurationStorage {

public static final Supplier<MessageConfiguration> START = () ->
MessageConfiguration.builder().build();
```

```
public static final Supplier<MessageConfiguration> ERROR = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().text("Щось пішло не так. Будь ласка,
спробуйте ще раз.").build()).build();

public static final Supplier<MessageConfiguration> CURRENCY_RATE = () ->
MessageConfiguration.builder().text(MessageTextConfiguration.builder().textGenerator(
RATE_TEXT).build()).build();

public static final Supplier<MessageConfiguration> START_WITHOUT_WALLET = () ->
MessageConfiguration.builder().keyboardType(KeyboardType.REPLY)
.text(MessageTextConfiguration.builder().message("Привіт, {0}.\nОбери
дію:").argGenerator(START_ARGS).build()).callbacks(List.of(
List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Створити гаманець").build())
.id("CREATE-WALLET").build(),
ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Імпортувати гаманець").build())
.id("IMPORT-WALLET").build()),
List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\uD83D\uDD0D Пошук акаунту").build())
.id("SEARCH-ACCOUNT").build(),ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\uD83D\uDD0D Пошук
транзакції").build()).id("SEARCH-TRANSACTION").build()),
List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Курс TRX до валют").build())
.id("CURRENCY_RATE").build()))).build();

public static final Supplier<MessageConfiguration> START_WITH_WALLET = () ->
MessageConfiguration.builder().keyboardType(KeyboardType.REPLY)
.text(MessageTextConfiguration.builder().message("Привіт, {0}.\nОбери
дію:").argGenerator(START_ARGS).build())
.callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Інформація про акаунт").build())
.id("CREATE-WALLET").build(), ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Історія транзакцій").build())
.id("IMPORT-WALLET").build()), List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\uD83D\uDD0D Пошук акаунту").build())
.id("SEARCH-ACCOUNT").build(), ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\uD83D\uDD0D Пошук
транзакції").build()).id("SEARCH-TRANSACTION").build()),
List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Створити транзакцію").build())
.id("NEW-TRANSACTION").build(), ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Курс TRX до валют").build())
.id("CURRENCY_RATE").build()))).build();

public static final Supplier<MessageConfiguration> ACCOUNT = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().textGenerator(SELECTED_ACCOUNT_TEXT).build()
).callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\u2b05\ufe0f Назад").build())
.id("BACK_TO_SEARCH").next(Keyboard.SEARCH_ACCOUNT).inputValidator(ADDRESS_VALIDATOR)
.action(SEARCH_ACCOUNT_ACTION).multiClickable(true).build()))).build();

public static final Supplier<MessageConfiguration> SEARCH_ACCOUNT = () ->
MessageConfiguration.builder().text(MessageTextConfiguration.builder().text("Введіть
адресу акаунта, щоб переглянути інформацію про акаунт. Приклад адреси:
TBcAmJyQ63EYP5DJqkxCJFXBz8RzgjqPAQ").build()).build();

public static final Supplier<MessageConfiguration> OWN_ACCOUNT = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().textGenerator(OWN_ACCOUNT_TEXT).build())
```

```
.callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Показати адресу").build())
.id("OWN-ACCOUNT-ADDRESS").next(Keyboard.OWN_ACCOUNT_ADDRESS).build(),
ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Експортувати гаманець").build())
.id("EXPORT-ACCOUNT").next(Keyboard.OWN_ACCOUNT_EXPORT)
.action(EXPORT_WALLET_ACTION).inputValidator(SKIP_VALIDATION_WITH_DELETE).build()),
List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("Видалити гаманець").build())
.id("REMOVE-ACCOUNT").next(Keyboard.OWN_ACCOUNT_REMOVE).action(REMOVE_WALLET_ACTION)
.inputValidator(SKIP_VALIDATION_WITH_DELETE).build()))).build();

public static final Supplier<MessageConfiguration> OWN_ACCOUNT_ADDRESS = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().message("{0}").argGenerator(OWN_ACCOUNT_ADDR
ESS_ARGS).build()).callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\u2b05\ufe0f Назад").build())
.id("BACK_TO_OWN_ADDRESS").next(Keyboard.OWN_ACCOUNT).build()))).build();

public static final Supplier<MessageConfiguration> OWN_ACCOUNT_EXPORT = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().text("Введіть пароль.").build())
.callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\u2b05\ufe0f Назад").build())
.id("BACK_TO_OWN_ADDRESS").next(Keyboard.OWN_ACCOUNT).build()))).build();

public static final Supplier<MessageConfiguration> OWN_ACCOUNT_REMOVE = () ->
MessageConfiguration.builder()
.text(MessageTextConfiguration.builder().text("Введіть пароль.").build())
.callbacks(List.of(List.of(ButtonConfiguration.builder()
.label(ButtonLabelConfiguration.builder().text("\u2b05\ufe0f Назад").build())
.id("BACK_TO_OWN_ADDRESS").next(Keyboard.OWN_ACCOUNT).build()))).build();}
```

Клас Keyboard

```
package edu.diploma.ms.bot.engine.constants;

import edu.diploma.ms.bot.engine.entity.config.MessageConfiguration;
import edu.diploma.ms.bot.engine.entity.config.MessageConfigurationStorage;
import lombok.Getter; import lombok.RequiredArgsConstructor;

import java.util.function.Supplier;

@Getter @RequiredArgsConstructor public enum Keyboard {

START(MessageConfigurationStorage.START),
ERROR(MessageConfigurationStorage.ERROR),

CURRENCY_RATE(MessageConfigurationStorage.CURRENCY_RATE),
START_WITHOUT_WALLET(MessageConfigurationStorage.START_WITHOUT_WALLET),
START_WITH_WALLET(MessageConfigurationStorage.START_WITH_WALLET),
ACCOUNT(MessageConfigurationStorage.ACCOUNT),

OWN_ACCOUNT(MessageConfigurationStorage.OWN_ACCOUNT),
OWN_ACCOUNT_ADDRESS(MessageConfigurationStorage.OWN_ACCOUNT_ADDRESS),
OWN_ACCOUNT_EXPORT(MessageConfigurationStorage.OWN_ACCOUNT_EXPORT),
OWN_ACCOUNT_REMOVE(MessageConfigurationStorage.OWN_ACCOUNT_REMOVE),

SEARCH_ACCOUNT(MessageConfigurationStorage.SEARCH_ACCOUNT),;
```

```
Private final Supplier<MessageConfiguration> configuration;}
```

Клас Action

```
package edu.diploma.ms.bot.engine.constants;

import edu.diploma.ms.bot.business.action.DefaultActions;
import edu.diploma.ms.bot.business.action.StartActions;
import edu.diploma.ms.bot.business.action.TronActions;
import edu.diploma.ms.bot.business.action.TronWalletActions;
import edu.diploma.ms.bot.business.action.util.PaginationDefaultActions;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import lombok.Getter;
import lombok.RequiredArgsConstructor; import java.util.function.Consumer;

@Getter@RequiredArgsConstructor
public enum Action {

START_ACTION(StartActions.START_ACTION),
SET_INPUT_DATA_IN_RELATED_FIELD_ACTION(DefaultActions.SET_INPUT_DATA_IN_RELATED_FIELD
_ACTION),
SET_INPUT_DATA_AS_DOUBLE_IN_RELATED_FIELD_ACTION(DefaultActions.SET_INPUT_DATA_AS_DOU
BLE_IN_RELATED_FIELD_ACTION),
SET_INPUT_DATA_AS_INTEGER_IN_RELATED_FIELD_ACTION(DefaultActions.SET_INPUT_DATA_AS_IN
TEGER_IN_RELATED_FIELD_ACTION),
SEARCH_ACCOUNT_ACTION(TronActions.SEARCH_ACCOUNT_ACTION),;
private final Consumer<ResponseMessageWrapper> action;}
```

Клас TronActions

```
package edu.diploma.ms.bot.business.action;

import edu.diploma.ms.bot.business.action.util.PaginationDefaultActions;
import edu.diploma.ms.bot.business.exception.integration.RemoteMsException;
import
edu.diploma.ms.bot.business.integration.core.communication.CoreCommunicationService;
import edu.diploma.ms.bot.business.integration.core.model.*;
import
edu.diploma.ms.bot.business.integration.tron.communication.TronCommunicationService;
import
edu.diploma.ms.bot.business.integration.tron.model.AccountResourcesTronResponse;
import edu.diploma.ms.bot.business.integration.tron.model.TransactionTronResponse;
import edu.diploma.ms.bot.business.storage.FreezeBalanceRateStorage;
import edu.diploma.ms.bot.engine.constants.Keyboard;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.UserSession;
import edu.diploma.ms.bot.engine.entity.config.MessageConfigurationStorage;
import edu.diploma.ms.bot.engine.services.MessageSender;
import edu.diploma.ms.bot.engine.utils.MessageFormatUtil;
import lombok.AccessLevel; import lombok.NoArgsConstructor;
import java.math.BigDecimal; import java.util.function.Consumer;
import static
edu.diploma.ms.bot.business.exception.ExceptionCode.FAILED_TRANSACTION_EXCEPTION;
import static edu.diploma.ms.bot.engine.utils.MessageFormatUtil.formatMessage;

@NoArgsConstructor(access = AccessLevel.PRIVATE) public final class TronActions {
public static final Consumer<ResponseMessageWrapper> SEARCH_ACCOUNT_ACTION = m -> {
```

```
TronCommunicationService communicationService =
m.getCtx().getBean(TronCommunicationService.class);
UserSession userSession = m.getUserSession();
AccountResourcesTronResponse accountResources =
communicationService.getAccountResources(userSession.getInputData());
userSession.setSelectedAccount(accountResources);
MessageConfigurationStorage.setKeyboardToSession(userSession, Keyboard.ACCOUNT);
userSession.getActiveButton().deactivate();};

public static final Consumer<ResponseMessageWrapper> SEARCH_OWN_ACCOUNT_ACTION = m ->
{TronCommunicationService communicationService =
m.getCtx().getBean(TronCommunicationService.class);
UserSession userSession = m.getUserSession();
MessageConfigurationStorage.setKeyboardToSession(userSession, Keyboard.OWN_ACCOUNT);
AccountResourcesTronResponse accountResources =
communicationService.getAccountResources(userSession.getActiveWalletAddress());
userSession.setSelectedAccount(accountResources);};

public static final Consumer<ResponseMessageWrapper> SEARCH_TRANSACTION_ACTION = m ->
{ TronCommunicationService communicationService =
m.getCtx().getBean(TronCommunicationService.class);
UserSession userSession = m.getUserSession();
TransactionTronResponse transaction =
communicationService.getTransaction(userSession.getInputData());
userSession.setSelectedTransaction(transaction);
MessageConfigurationStorage.setKeyboardToSession(userSession, Keyboard.TRANSACTION);
        userSession.getActiveButton().deactivate();
    };

public static final Consumer<ResponseMessageWrapper>
SELECT_TRANSACTION_HISTORY_ACTION = m -> {
UserSession userSession = m.getUserSession();
PaginationDefaultActions.SELECT_PAGINATION_VALUE_ACTION.accept(m);
TransactionTronResponse selectedPaginationValue = (TransactionTronResponse)
userSession.getPaginationState().getSelectedPaginationValue();
TronCommunicationService communicationService =
m.getCtx().getBean(TronCommunicationService.class);
TransactionTronResponse transaction =
communicationService.getTransaction(selectedPaginationValue.getHash());
userSession.setSelectedTransaction(transaction);
userSession.getActiveButton().deactivate();};

public static final Consumer<ResponseMessageWrapper> PREPARE_TRANSFER_CURRENCY_ACTION
= m -> {UserSession userSession = m.getUserSession();
if (userSession.getCurrencyTransferCoreRequest() == null) {
CurrencyTransferCoreRequest request = CurrencyTransferCoreRequest.builder()
.telegramUserId(userSession.getTelegramUserInfo().getUserId()).build();
userSession.setCurrencyTransferCoreRequest(request);}};

public static final Consumer<ResponseMessageWrapper>
CALCULATE_TRANSFER_CURRENCY_ACTION = m -> {UserSession userSession =
m.getUserSession();
userSession.getActiveButton().deactivate();
CoreCommunicationService coreCommunicationService =
m.getCtx().getBean(CoreCommunicationService.class);
BandwidthPriceCoreResponse bandwidthPriceCoreResponse = coreCommunicationService

.transferCurrencyPrice(userSession.getCurrencyTransferCoreRequest());
userSession.getMessageConfiguration().getText()

.setText(formatMessage("Вартість переказу: {0} одиниць пропускної
здатності.\nЗверніть увагу, що за кожну одиницю пропускної здатності," +
" якої недостатньо для транзакції, з акаунту буде спалено 0.00004 TRX.\nВиконати
переказ?", bandwidthPriceCoreResponse.getBandwidthPrice()));};
```

```
public static final Consumer<ResponseMessageWrapper> CALCULATE_FREEZE_BALANCE_ACTION
= m -> {
UserSession userSession = m.getUserSession();
FreezeBalanceCoreRequest request = FreezeBalanceCoreRequest.builder()
.telegramUserId(userSession.getTelegramUserInfo().getUserId())
.amount(Integer.parseInt(userSession.getInputData())).build();
userSession.setFreezeBalanceCoreRequest(request);

userSession.getActiveButton().deactivate();
CoreCommunicationService coreCommunicationService =
m.getCtx().getBean(CoreCommunicationService.class);
BandwidthPriceCoreResponse bandwidthPriceCoreResponse = coreCommunicationService
.freezeBalancePrice(request);

FreezeBalanceRateStorage freezeBalanceRateStorage =
m.getCtx().getBean(FreezeBalanceRateStorage.class);
double bandwidthToTrxRate = freezeBalanceRateStorage.getBandwidthToTrxRate();
BigDecimal receivedBandwidth = BigDecimal.valueOf(bandwidthToTrxRate)
.multiply(BigDecimal.valueOf(request.getAmount()));
MessageConfigurationStorage.setKeyboardToSession(userSession,
Keyboard.FREEZE_CONFIRM);

userSession.getMessageConfiguration().getText().setText(formatMessage("Вартість
застави коштів: {0} одиниць пропускної здатності.\nОтримаєте балів пропускної
здатності: {1} \nЗверніть увагу, що за кожну одиницю пропускної здатності, якої
недостатньо для транзакції, з акаунту буде спалено 0.00004 TRX.\nВиконати переказ?",
bandwidthPriceCoreResponse.getBandwidthPrice(),
receivedBandwidth.stripTrailingZeros().toPlainString()));};}
```

Клас TronHistoryTextGenerator

```
package edu.diploma.ms.bot.business.text.button;

import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.pagination.EntityForPagination;
import lombok.AccessLevel;
import lombok.NoArgsConstructor;
import java.util.function.BiFunction;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public final class TradeHistoryTextGenerator {
public static final BiFunction<ResponseMessageWrapper, EntityForPagination, String>
ORDER_PAGINATION_BUTTON_TEXT_GENERATOR = (m, entity) ->
(String) entity.getButtonTextParams()[0];}
```

Клас TronAccountTextGenerator

```
package edu.diploma.ms.bot.business.text.message;

import
edu.diploma.ms.bot.business.integration.tron.model.AccountResourcesTronResponse;
import edu.diploma.ms.bot.business.storage.CurrencyRatesStorage;
import edu.diploma.ms.bot.engine.entity.ResponseMessageWrapper;
import edu.diploma.ms.bot.engine.entity.UserSession;
import lombok.AccessLevel;
import lombok.NoArgsConstructor;
import org.springframework.context.ApplicationContext;
```

```
import java.math.BigDecimal;
import java.util.function.Function;

import static edu.diploma.ms.bot.business.utils.NumberConverter.toPlainString;
import static edu.diploma.ms.bot.engine.utils.MessageFormatUtil.formatMessage;

@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class TronAccountTextGenerator {

public static final Function<ResponseMessageWrapper, String> SELECTED_ACCOUNT_TEXT =
m -> {UserSession userSession = m.getUserSession();
AccountResourcesTronResponse selectedAccount = userSession.getSelectedAccount();
if (selectedAccount.getAddress() != null && selectedAccount.getBandwidthLimit() != 0)
{ String accountMessage = formatMessage("Інформація про акаунт:\nАдреса:
{0}\nДоступний баланс (TRX): {1}\nПропускна здатність: вільно {2} з {3}\nБаланс під
заставою (TRX): {4}\n", selectedAccount.getAccountMessageArgs());
String rateMessage = getTronRateMessage(m.getCtx());
String accountRateMessage = getTronAccountRateMessage(selectedAccount, m.getCtx());
return accountMessage.concat(rateMessage).concat(accountRateMessage);
} else if (selectedAccount.getExists()) {
return "Вказаний акаунт не активований. Для активації, необхідно зробити переказ на
акаунт (додатково буде знято 0.1 TRX)";
} else {return "Вказаного акаунта не існує.";}};

public static final Function<ResponseMessageWrapper, String> OWN_ACCOUNT_TEXT = m ->
{UserSession userSession = m.getUserSession();
AccountResourcesTronResponse selectedAccount = userSession.getSelectedAccount();
if (selectedAccount.getAddress() != null && selectedAccount.getBandwidthLimit() != 0)
{ String accountMessage = formatMessage("Інформація про акаунт:\nАдреса:
{0}\nДоступний баланс (TRX): {1}\n Пропускна здатність: вільно {2} з {3}\nБаланс під
заставою (TRX): {4}\n", selectedAccount.getAccountMessageArgs());
String rateMessage = getTronRateMessage(m.getCtx());
String accountRateMessage = getTronAccountRateMessage(selectedAccount, m.getCtx());
return accountMessage.concat(rateMessage).concat(accountRateMessage);
} else if (selectedAccount.getExists()) {
return formatMessage("Ваш акаунт з адресою {0} не активований. Для активації,
необхідно зробити переказ на акаунт (додатково буде знято 0.1 TRX)",
userSession.getActiveWalletAddress());
} else {return "Вказаного акаунта не існує.";}}; public static final
Function<ResponseMessageWrapper, String> RATE_TEXT = m ->
getTronRateMessage(m.getCtx());

private static String getTronRateMessage(ApplicationContext ctx) {
CurrencyRatesStorage currencyRatesStorage = ctx.getBean(CurrencyRatesStorage.class);
return formatMessage("Курс 1 TRX до гривні: {0}\nКурс 1 TRX до долара: {1}\n",
toPlainString(currencyRatesStorage.getUahRate()),
toPlainString(currencyRatesStorage.getUsdRate()));}

private static String getTronAccountRateMessage(AccountResourcesTronResponse account,
ApplicationContext ctx) {
CurrencyRatesStorage currencyRatesStorage = ctx.getBean(CurrencyRatesStorage.class);
if (account.getBalance() == 0.0) {
return formatMessage("Доступний баланс TRX в гривні: {0}\nДоступний баланс TRX в
доларах: {1}\n", "0", "0");}
BigDecimal balance = BigDecimal.valueOf(account.getBalance());
BigDecimal balanceUah =
balance.multiply(BigDecimal.valueOf(currencyRatesStorage.getUahRate()));
BigDecimal balanceUsd =
balance.multiply(BigDecimal.valueOf(currencyRatesStorage.getUsdRate()));
return formatMessage("Доступний баланс TRX в гривні: {0}\nДоступний баланс TRX в
доларах: {1}\n", balanceUah.stripTrailingZeros().toPlainString(),
balanceUsd.stripTrailingZeros().toPlainString());}}
```

ДОДАТОК Г

(Обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота

Виконав: студент IV курсу, група ІПЗ-17-1 Мельник Андрій
Керівник: доцент, кандидат технічних наук Праворська Н.І.

Мета дослідження	Проектування та імплементація електронного гаманця для криптовалюти Tronix з інтерфейсом у вигляді Telegram бота, який би вирішував наявні проблеми сучасних рішень.
Об’єкт дослідження	Процеси функціонування електронних гаманців для криптовалют, блокчейн криптовалюти Tronix, технологія розробки Telegram ботів.
Предмет дослідження	Моделі та механізми створення електронних гаманців для криптовалют та Telegram ботів.
Завдання дослідження	Провести аналіз специфіки роботи електронних гаманців, технології блокчейн, криптовалюти Tronix та Telegram ботів. Дослідити наявне програмно-технічне забезпечення предметної області. Визначити вимоги до електронного гаманця та функцій, які він має виконувати. Виконати проектування програмної системи, інтерфейсу користувача та обрати технології і методи реалізації системи. Виконати програмну реалізацію прийнятих рішень. Провести тестування розробленої системи. Проаналізувати отримані результати та сформувати рекомендації щодо напрямів продовження роботи

Актуальність теми

Особливості TRON, Tronix

делеговане підтвердження частки (DPoS) – висока швидкість переказів
модель ресурсів, пропускна здатність (Bandwidth) – безкоштовні перекази
платформа для децентралізованих додатків (DApps)

Особливості Telegram в ролі платформи для додатків

відсутність доступу з зовнішнього світу
ідентичний інтерфейс для всіх гаджетів
одночасне оновлення функціоналу для всіх платформ

3.3-7 Транзакцій на секунду

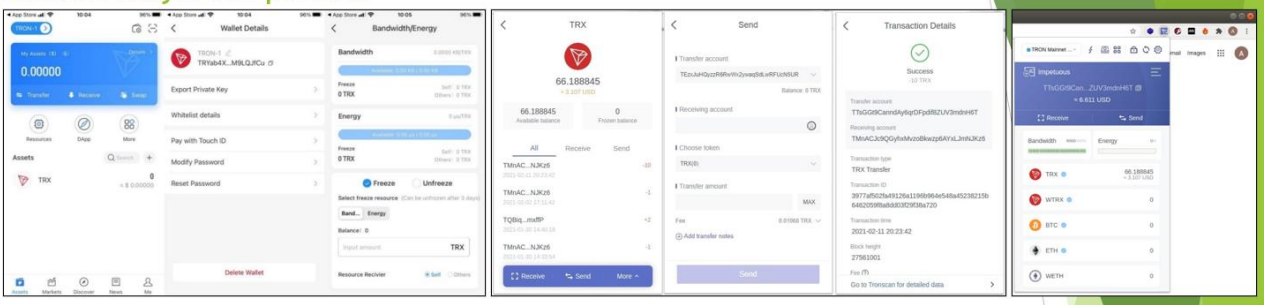
до 700 Транзакцій на секунду

15-20 Транзакцій на секунду

Актуальність теми

Актуальність теми полягає у відсутності реалізації електронного гаманця для Tronix, який був би доступний в єдиному вигляді одразу на всіх основних комп'ютерних платформах у різних виглядах, таких як: додаток для комп'ютера чи смартфону під різними операційними системами, веб-додаток з доступом через браузер тощо.

Аналіз існуючих рішень

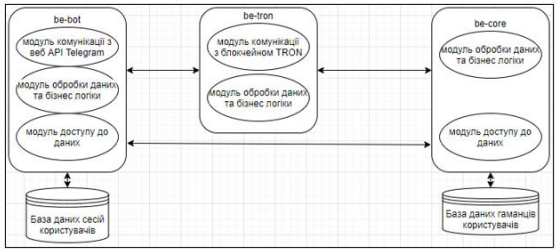


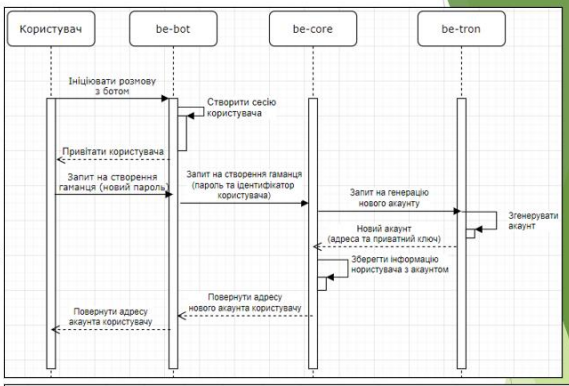
TokenPocket

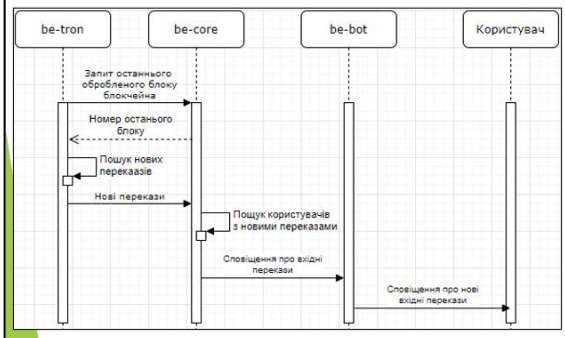
TronLink

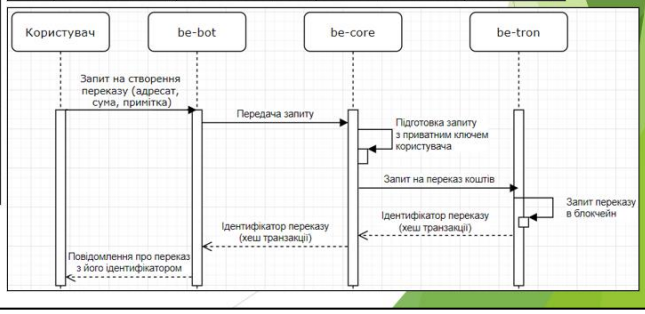
Недоліки:
1. Відсутність комп’ютерних та повноцінних веб-версій
2. Відсутність сповіщень про вхідні перекази
3. Відсутність єдиного інтерфейсу

Проектування



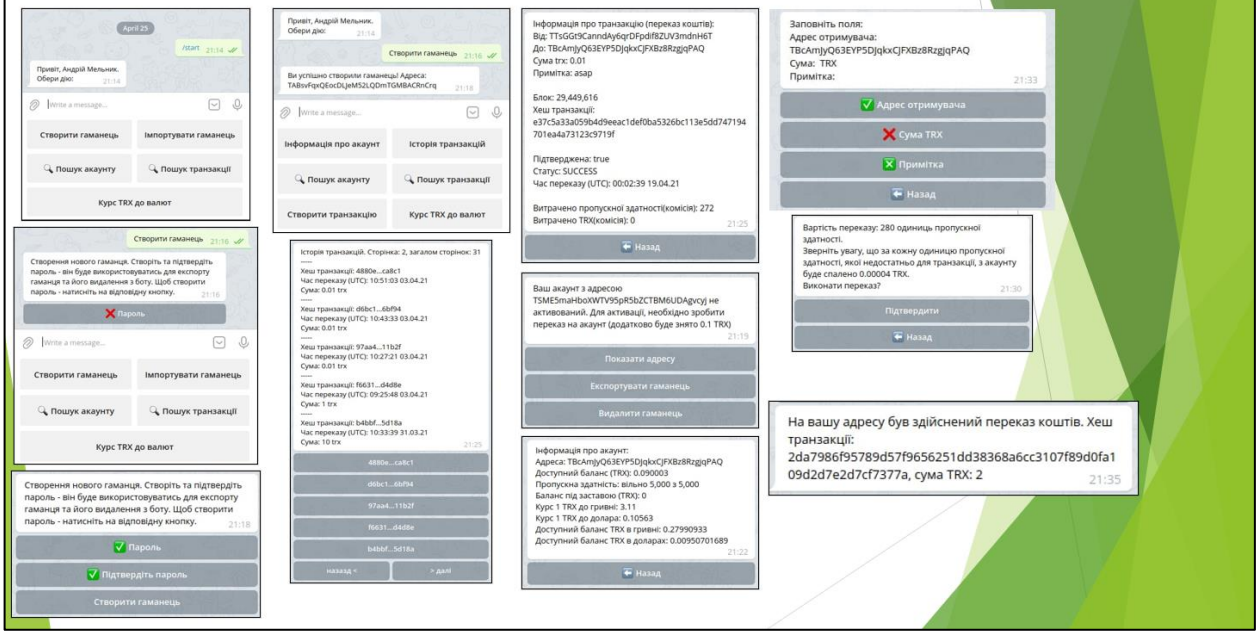






Інструменти та технології

Результати розробки



Отримані результати

Розроблена система дозволяє:
- створювати та імпортувати існуючі гаманці;
- створювати транзакції з переказами коштів, заставою та відміною застав;
- переглядати баланси (як свій, так і інших користувачів), курс валюти;
- переглядати історію транзакцій, пошук транзакції;
- отримувати сповіщення про вхідні перекази.

Практична цінність

Практична цінність отриманих результатів полягає в успішній розробці електронного гаманця для криптовалюти Tronix, який можна використовувати на всіх традиційних платформах з єдиним інтерфейсом. Завдяки даній особливості реалізації, розроблена програмна система має високу конкурентну спроможність в порівнянні з існуючими рішеннями.

Висновки

В результаті виконання дипломного проекту було проведено аналіз криптовалюти Tronix та її особливостей, Telegram як платформи для розробки, визначено недоліки існуючих рішень. Запропоновано рішення, яке дозволять вирішити існуючі проблеми на ринку. Було спроектовано та реалізовано систему, яка повністю відповідає визначеними функціональним вимогам.

Серед можливих напрямів продовження роботи над проектом варто відмітити наступні варіанти: система обміну валютами, система для торгівлі валютами, реалізація платіжної системи тощо.

Завідувачу кафедри
канд. техн. наук, доцент.
Бедратюку Л. П.

здобувача вищої освіти
ФПКТС, 4 курсу, групи ІПЗ-17-1
Мельника Андрія Васильовича

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

2 червня 2024р.
дата

підпис

Tue Jun 01 07:24:15 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 23%**

	Документ		Сумарний збіг по Базі Даних	
ID: 91747 Назва: Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота Додано в БД: 2021-06-01 Автора: А. В. Мельник Керівники: Н. І. Праворська Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	164446	2264	5493 (3%)	72 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008134723

Дата перевірки:
02.06.2021 10:21:22 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2021 10:25:12 EEST

ID користувача:
100005589

Назва документа: **Диплом Мельник ІПЗ-17-1 v5**

Кількість сторінок: **149** Кількість слів: **30046** Кількість символів: **271570** Розмір файлу: **3.66 MB** ID файлу: **1008215667**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.62%
Схожість

Найбільша схожість: **1.75%** з джерелом з Бібліотеки (ID файлу: **1008215685**)

4.17% Джерела з Інтернету	542	Сторінка 151
1.9% Джерела з Бібліотеки	62	Сторінка 157

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	68
Підозріле форматування	25 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Мельник Андрій Васильович

Тема Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 117

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті проаналізовано предметну область, існуючі рішення на ринку, визначені їх недоліки. Було визначено функціональні вимоги до системи. Була побудована архітектура додатку, проаналізовані та обрана база даних, обрані інструменти для реалізації проекту. Була створена програмна система, проведене тестування програмного забезпечення і сформовані висновки та напрямки для подальшого продовження та розвитку проекту.

2. Висновок про відповідність проекту поставленому завданню Дипломна робота виконана у відповідності до завдання із дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі була обгрунтована актуальність теми, сформульована мета та завдання роботи. У першому розділі проаналізовано та вивчено роботу технології блокчейн, особливості криптовалюти Tronix, проаналізовано існуючі рішення та визначені виконана розгорнута постановка задачі. У другому розділі було визначено, що система буде відповідати моделі клієнт-сервер, було проаналізовано різні серверні архітектури, в результаті чого було обрано мікросервісну архітектуру. Був обраний нереляційний тип баз даних, мова програмування та сучасні фреймворки в якості інструментів для розробки. У третьому розділі було реалізовано програмну систему, створена інструкція для користувача, визначені системні вимоги для роботи програми, створена інструкція для запуску системи. В четвертому розділі було обрано актуальні методи тестування, створені тестові сценарії та проаналізовано результати тестування.

4. Позитивні сторони проекту Тематика роботи є актуальною. У повній мірі проаналізовано роботу технології блокчейн та особливості обраної криптовалюти. Визначено переваги платформи Telegram з точки зору розробки програм, вирішено реальні проблеми існуючих рішень, застосовано сучасні технології та архітектурні підходи при розробці системи.

5. Негативні сторони проекту У роботі розглянуто та проаналізовано декілька різних алгоритмів роботи криптовалют – краще було б проаналізувати алгоритм криптовалюти Tronix більш детально. Недолік не зменшує позитивне враження від роботи.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення у вигляді рисунків та діаграм виконане відповідно до теми дипломного проекту. Оформлення виконано з дотриманням всіх вимог. Пояснювальна записка відповідає вимогам до її оформлення.

7. Відгук про дипломний проект в цілому Робота заслуговує позитивної оцінки. Матеріал роботи структурований, чіткий, послідовний та в повній мірі розкриває обрану тему, що дозволяє чітко розуміти викладений матеріал у рамках тематики роботи. Графічний матеріал дає змогу наочно побачити особливості та переваги рішень, прийнятих для вирішення поставленої задачі.

8. Інші зауваження

9. Оцінка дипломного проекту Робота виконана в повному обсязі. З огляду на зазначені позитивні та негативні сторони проекту, робота заслуговує на оцінку «відмінно» (4,75/A).

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи)
Лисенко Сергій Миколайович, доктор технічних наук, доцент кафедри комп'ютерної інженерії та системного програмування (КІСП) ХНУ

"02" червня 2021 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Програмне забезпечення для реалізації електронного гаманця криптовалюти Tronix з реалізацією інтерфейсу у вигляді Telegram-бота»

Автор: Мельник Андрій Васильович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Науковий керівник: Праворська Наталія Іванівна, канд. пед. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломного проекту системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень в бланках (титулка, бланк завдання, в структурі підрозділів ВСТУПУ) та в назвах публікацій джерел;
2) В якості запозичень системою було зафіксовано послідовність вихідного коду, які є спільними для великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) усі запозичення фрагментарні, або мають належним чином оформленні посилання.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності / схожості, складає 6,07% і адресується до 604 першоджерел, що з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ Н.І. Праворська

Гарант ОП _____ Л.П. Бедратюк

Завідувач кафедри _____ Л.П. Бедратюк