Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

БАГ-БАУНТІ ПЛАТФОРМА НА БЛОКЧЕЙНІ

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи

Гороховський К.С

Виконав студент

Будаєв А.Ю.

Київ 2021

Анотація

У роботі розглянуто поняття блокчейну, смарт-контрактів та мета їх застосування, проблеми та особливості структури застосунків з використанням блокчейну. Також розглянуто практичне застосування блокчейну у застосунку, а саме платформи для проведення баг-баунті.

Вступ

На сьогоднішній день тема блокчейн набуває все більшої популярності. Частково цим можна завдячити криптовалютам, які б'ють рекорди по дохідності для інвесторів останніми роками. Але чи можна блокчейн однозначно ототожнювати з криптовалютами? Насправді, криптовалюти – часткове застосування блокчейну, який також має багато інших сфер для застовування. Все більше сервісів починають використовувати блокчейн: OpenBazaar - площадка для інтернет-торгівлі, Storj – сервіс для розподіленого безпечного зберігання даних, Ripple – сервіс для грошових переказів, проект Aave – платформа для отримання та надання кредитів тощо. Новизна технології, нестача досвіду у розробників – це відкрита сфера для досліджень.

Все більше і більше розробників намагаються знайти шляхи вирішення проблем кібератак на їх сервіси, оскільки наслідки можуть бути колосальними. Так, у 2020 році у мережу злили дані 267 млн. користувачів Facebook, наслідками чого став штраф 5 млрд. дол. США. У 2016 році хакери за допомогою зловмисницького ПЗ здійснили втручання у вибори США з метою підробки даних у голосуванні. Наслідок – падіння рейтингів Трампа.

Зрозуміло, що дешевше виявляти вразливості на етапі розробки, ніж потім нести багатомільярдні збитки. Також необхідно пустити у мирне русло

хакерів, що б вони займалися улюбленою справою без ризику сісти за грати. Для цього були створені платформи, де хакери за гроші перевіряють на вразливості застосунки розробників. У даній роботі досліджено, як блокчейн може вплинути на процес баг-баунті.

Зміст

Розділ 1. Технологія блокчейн та смарт-контракти

1.1 Ланцюжок блоків

Блокчейн – це структура зберігання даних, який можна ототожнити зі зв'язним списком, де кожний елемент – блок – зберігає хеш-функцію від вмісту попереднього блоку. Хеш – це дані, результат виконання шифрувальної функції після обробки вхідних даних. Хеш зручно використовувати, коли дані необхідно звести до одного уніфікованого розміру, бо які б два різних за розміром рядка не були передані, на виході отримується два хеша одного розміру. Одні і ті ж вхідні дані будуть однозначно зашифровуватись у однаковий хеш. Якщо у даних змінити хоча б один символ, зміниться і хеш. Це і забезпечує непідробність даних у блокчейні після запису у блок, оскільки якщо в попередньому блоці щось змінити, то в наступному блоці хеш, згенерований з даних попереднього, буде вже не співпадати з фактичним.

Блокчейн – децентралізована система, що значить, що нема центрального серверу. Хто ж забезпечує її роботу і з якою мотивацією? «Забезпечення роботи» - це генерування наступного блоку з даними на основі попереднього, тобто обрахунок хешу, а також перевірку валідності транзакцій за певним алгоритмом. Він називається алгоритмом консенсусу, який вважається досягнутим, коли 51% учасників мережі будуть вважати транзакцію валідною. Мотивація ж полягає у нагороді перевіряльників транзакцій криптовалютою.

Два основних алгоритми консенсусу – так звані PoW та PoS:

- PoW (Proof of Work) - «доказ виконаної роботи» - консенсус полягає у перевірці валідності вже згенерованих фактичним, а це складні криптографічні обрахунки. Тобто ті учасники мережі, які віддали свої

обчислювальні потужності для цих розрахунків отримують «видобувають» за це криптовалюту.

- PoS (Proof of Stake) – «доказ володіння частки» - полягає у тому, що учасник вносить свої монети на зберігання на певний термін і отримує за це нагороду. Користь від цієї дії в тому, що монет на руках залишається менше – а значить менше вірогідність саботування системи за допомогою 51% монет.

1.2. Цифровий підпис

Електронний цифровий підпис – набір байтів, що дозволяє підтвердити власність певних даних певній особі. ЕЦП отриманих шляхом криптографічного перетворення даних, що однозначно ідентифікує користувача (закритий ключ), та даних, що підлягають підпису. Отже, підпис пов’язаний як з автором, так і з документом і будь-які зміни підписаних даних або користувача приводитимуть до зміни підпису.

З огляду на те, що підписані дані можуть різного розміру, то для певної уніфікації доцільно накладати ЕЦП на хеш від даних. Оскільки хеш однозначно відповідає даним, то і підпис, накладений на хеш, однозначно відповідатиме даним. Крім того, алгоритми хешування працюють швидше алгоритмів ЕЦП, тому немає необхідності шифрувати алгоритмами ЕЦП великий об’єм даних при тому, що можна зашифрувати хеш і отримати те саме підтвердження авторства та незмінності документу.

На сьогодні найбільш широке застосування отримав алгоритм асиметричного шифрування з відкритим ключем, який полягає у наступному:

- Генерується пара закритого та відкритого ключа таким чином, що вони однозначно відповідають один одному, але знаючи відкритий ключ, немає змоги вирахувати закритий. Відкритий, або публічний ключ можуть знати всі, а закритий ключ слід знати тільки власнику.

Однозначна відповідність полягає у тому, що с різним закритим ключам не може відповідати один і той же відкритий

- Алгоритми криптографічного шифрування дозволяють зашифрувати дані закритим ключем так, щоб можна було успішно розшифрувати його відкритим ключем.
- Також шифрування закритим ключем відбувається таким чином, що розшифровка даних відкритим ключем означає, що початкові дані були незміненими і були зашифровані виключно тим закритим ключем, який відповідає відкритому.

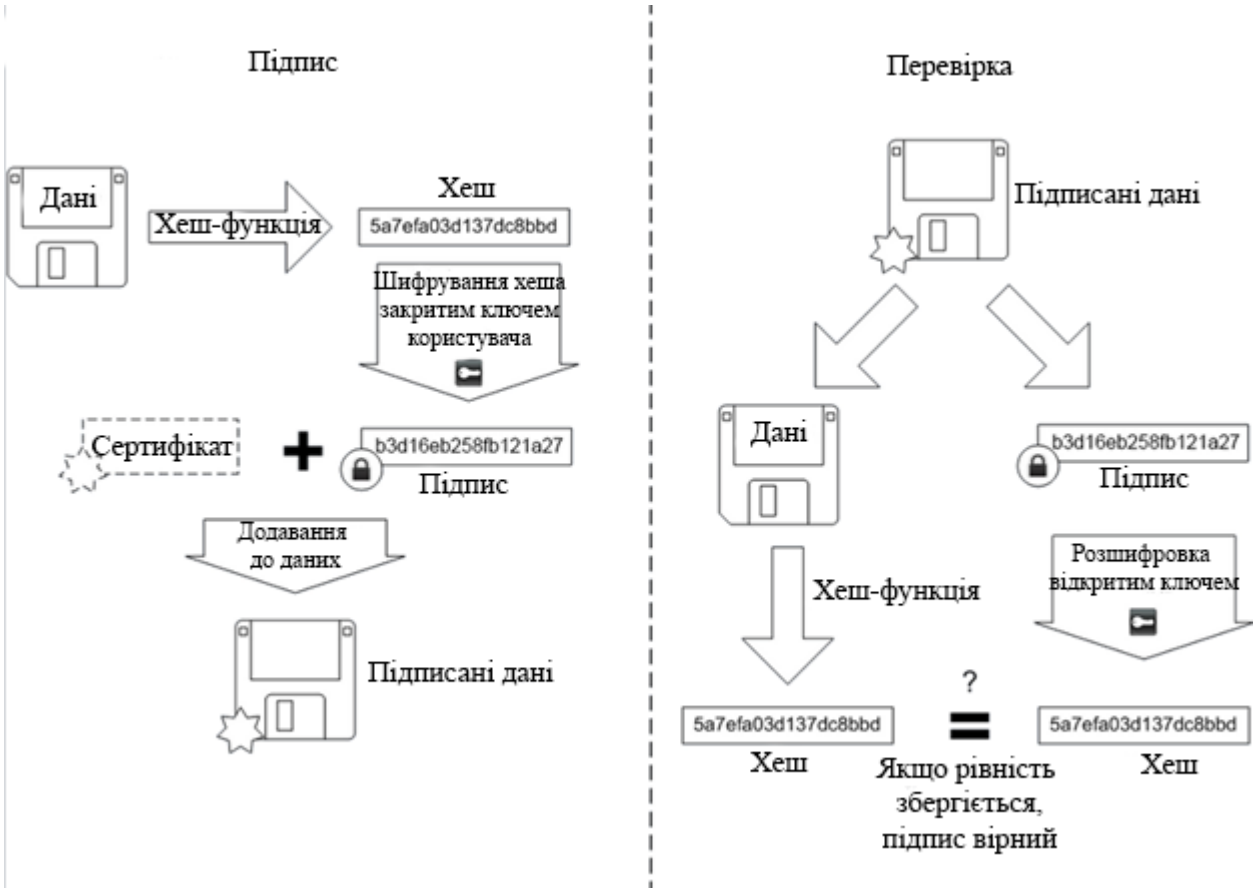


Рис.1.2.1 Робота алгоритмів ЕЦП

Як зазначено у розділі 1.1, за валідацію транзакцій відповідальні майнери, але що значить провалідувати транзакцію? У основі валідації і лежить саме алгоритм асинхронного ЕЦП, тобто ініціатор транзакції «підписує» її своїм лише йому відомомим закритим ключем. Важливо розуміти, що шифрування

даних проходить не задля анонімності даних, а задля генерації ЕЦП, яка по суті є контрольною сумою. Напроти, дані відкриті, просто до них додається ЕЦП, щоб валідатори змогли перевірити доступним всім відкритим ключем підписанта. При розшифровці підпису звіряється початкові дані з розшифрованим підписом, якщо вони не співпадають, значить, дані не дійсні та/або підписані не тим закритим ключем. Інакше транзакція відхиляється. Саме тому будь-яка взаємодія у блокчейні дає впевненість, що іде взаємодія саме з тією людиною, за яку вона себе видає.

1.3. Смарт-контракти на блокчейні

Саме поняття смарт-контрактів не нове і було запропоноване ще до блокчейну, але на сьогоднішній день розповсюджені звичайні контракти. Так, за певне невиконання пунктів звичайного контракту на сторону можуть бути накладені санкції (за невчасне виконання робіт – штраф тощо), але саме виконання робіт не гарантується. Регулювання контракту здійснюється третіми сторонами, державою наприклад, яка за допомогою механізму суду та нормативно-правовою регуляцією змушує сторони до певних дій.

Смарт-контракти на блокчейні покликані зняти питання можливого невиконання обов’язків сторонами угоди, тому що всі умови заздалегідь прописані у коді та розміщені у блокчейні. Блокчейн висупає у ролі децентралізованого неупередженого регулятора, що забезпечує автоматизоване справедливе розподілення цінностей (гроші, активи) та спосіб їх розподілення в залежності від виконаних умов. Це зменшує вплив фактору довіри між сторонами угоди. А як зазначено у розділі 1.2., всі дані, що були додані до блокчейну є незмінними, що убезпечує контракт від небажаних змін. Так само простіше проводити аудит контрактів, оскільки всі бажаючі зможуть перевірити сховище контракту, його баланс та пов’язані з ним транзакції, тобто перевірити приналежність житла, автомобіля, активів тощо легко було б перевірити у суді, наприклад, але поки що смарт-контракти не мають під собою нормативно-правової сили.

При написанні контрактів важливо визначити три речі:

- Вхідний параметр – це інтерфейс взаємодії з контрактом, які параметри він може приймати на вхід, тобто, кажучи простою мовою, функції, які може викликати користувач та вхідні параметри, які він може їй передавати
- Сховище – ті дані будуть зберігатися в смарт-контракті
- Код, який оброблятиме вхідний параметр, тобто ядро контракту. Він забезпечує виконання певних дій та зміну стану контракту в залежності від вхідного параметру.

```
parameter int;
storage   int;
code
      {
        CAR;         # @parameter
        NIL operation; # list operation : @parameter
        PAIR;        # pair (list operation) @parameter
      };
```

Рис 1.3.1

Розділ 2. Вирішення проблем сучасних сервісів за допомогою смарт-контрактів

2.1 Наявні проблеми

Якщо розглядати сучасний процес укладання угод між сторонами без смарт-контрактів, то маємо низку проблем:

А) Присутній фактор недовіри – при здійсненні купівлі в інтернет магазині так чи інакше одна зі сторін, покупець або продавець, здійснює «крок вперед». Це означає, що в більшості випадків покупець спочатку сплачує за товар, а потім чекає на його доставку. Таким чином продавець «перестраховується» задля гарантованого отримання сплати. Але від особи покупця, зрозуміло, все виглядає дуже ненадійно. Це відбувається тому, що

між покупцем і продавцем є відстань і неможливо обміняти товар на гроші інакше.

Аналогічна ситуація на прикладі сервісу для віддаленого бронювання квартир Airbnb – подорожуючий перший сплачує за квартиру, не знаючи достовірно, чи існує вона взагалі. Тобто все зводиться до фактору довіри – люди вибирають продавця з найкращими відгуками и власника квартир з найкращим рейтингом – а це вимушена довіра до якості формування рейтингу та до досвіду попередніх клієнтів.

Б) Зайва витрата часу – сучасне укладання угод не є безпомилковим на швидким. Так, оформлення кредиту, продаж/купівля нерухомості, будь-які інші контракти містять у собі людський фактор у особі рієлтора, банка або ж нотаріуса, яким властиво помилятися. Якщо умови, укладені на паперовому контракті однією стороною не виконані – у іншої сторони з’являється необхідність бігати по судам і доводити свою правоту, що є додатковий час.

В) Додаткові грошові витрати – вищеописані посередники (рієлтори, банки тощо) потребують оплати послуг.

Г) Централізація та регулювання з боку третіх осіб (держави, суддів тощо). Жодна транзакція у сучасних платіжних системах не відбувається без того, щоб про неї не знала держава. Але вона не тільки знає, але й дозволяє собі її заблокувати, якщо їх щось не сподобається (походження грошей, призначення платежу) через вплив на банки.

Д) Ненадійність баз даних або архівів, у яких зберігаються контракти (бази даних уразливі до атак, архіви до пожеж).

2.2. Можливі рішення

Звісно, що вищеописані проблеми мають наразі часткові вирішення – OLX має безпечну доставку, у таксі поїздка закінчується коли геолокація співпадає з місцем призначення, захист прав споживачів на законодавчому

рівні і т.д., але фактор доброчесності OLX, сервісу таксі, суда залишається. Вони не є настільки ж ефективними як рішення, які пропонують смарт-контракти.

Оскільки розумні контракти гарантують виконання прописаних раніше умов та повну невідворотність заданих дій, у сторін угоди зникає потреба довіряти один одному задля виконання угоди. Хід думок людини змінюється з «я виберу оцього виробника тому, що в нього високий рейтинг» на «я виберу цього виробника тому, що в нього є те, що мені треба». Для оплати товару створюється смарт контракт, у якому прописані всі необхідні умови. Спрощено – «поки контракт не буде поповнено на необхідну суму доставка товару не здійснюється. Якщо контракт поповнений покупцем, кошти не переходять продавцю, поки покупець не отримає товар».

Аналогічний випадок, коли N партнерів вирішують зробити спільну інвестицію. Створюється смарт контракт, кожен з N партнерів поповнює його на задану суму. У контракті є умова, що будь-яка дія з коштами може бути виконана виключно за згоди (підписами) всіх поповнювачів.

За рахунок гнучкості прописаної логіки розумні контракти будуть корисні у більш складних випадках, коли необхідно перевірити наявність квартири у державному реєстрі нерухомості та контролювати код до замка (до прикладу з Airbnb). Тобто як оренда, так і продаж нерухомості можуть здійснюватися простими діями з боку сторін угоди без посередників.

На рисунку 2.2.1 видно, як з одного боку власники готелів розміщують дані про номери (ціна, кількість місць тощо) у блокчейні, а з іншого боку туристи бачать їх та бронюють (використовуючи смарт-контракти) в обхід таких посередників як Airbnb, які брали б за це комісію.

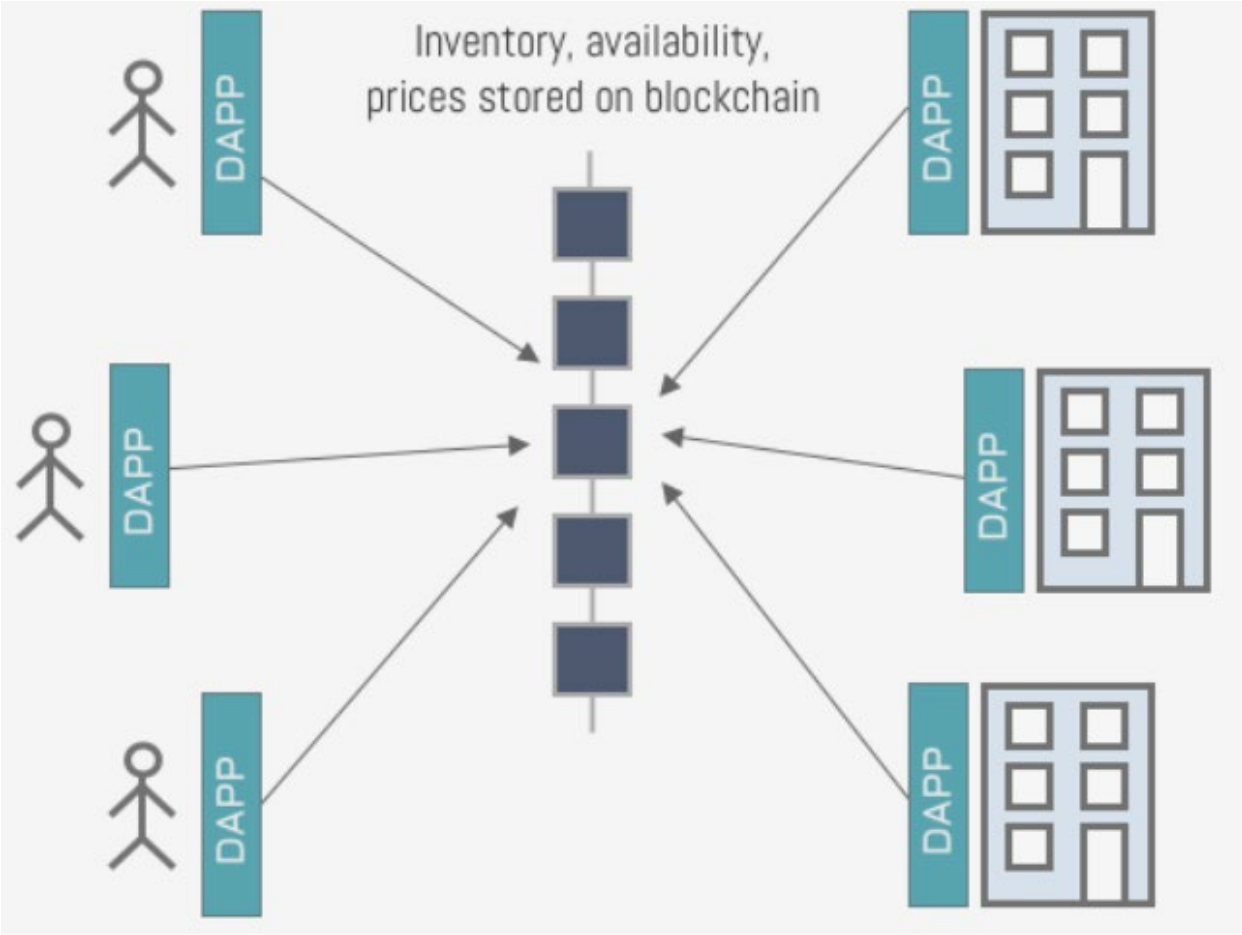


Рис 2.2.1

Безпомилкове автоматизоване виконання коду здатне прискорити виконання процесів та гарантуватиме саме таку логіку, яку запрограмував власник контракту. Якщо умови для зняття коштів не виконаються, з ними нічого не трапиться і незаплановано вони нікуди не дінуться.

Багаторазове дублювання документів та зашифрований спосіб їх зберігання, а також сама логіка взаємодії людей з контрактом (через шифрування ключами) робить роботу з документами та їх зберігання безпечним. В залежності від консенсусу блокчейну, підробити транзакцію або навіть в теорії неможливо, або для цього треба умови, які на практиці ніколи не виконаються. Якщо в майбутньому смарт-контракти набудуть нормативно-правової сили, то перевірити зміст контракту та дії з ним у суді стане легше, адже він буде «увіковічений» у блокчейні та гарантовано не підроблений.

Розділ 3. Підходи до реалізації макроструктури застосунків на блокчейні

3.1. Розподіл функцій між клієнтом, сервером та блокчейном

Через новизну технології, її складності та надмірного використання ресурсів, недостатньої кількості розробників та необізнаності аудиторії ставити блокчейн в один ряд з класичним підходом по зберіганню даних по швидкодії, ресурсовитратності неможна. Так, на рисунку 3.1.1 бачимо графік медіани часу підтвердження транзакцій у мережі Bitcoin за даними blockchain.com, а на рисунку 3.1.2 графік медіани комісії за транзакцію у мережі Ethereum за даними blockchair.com.

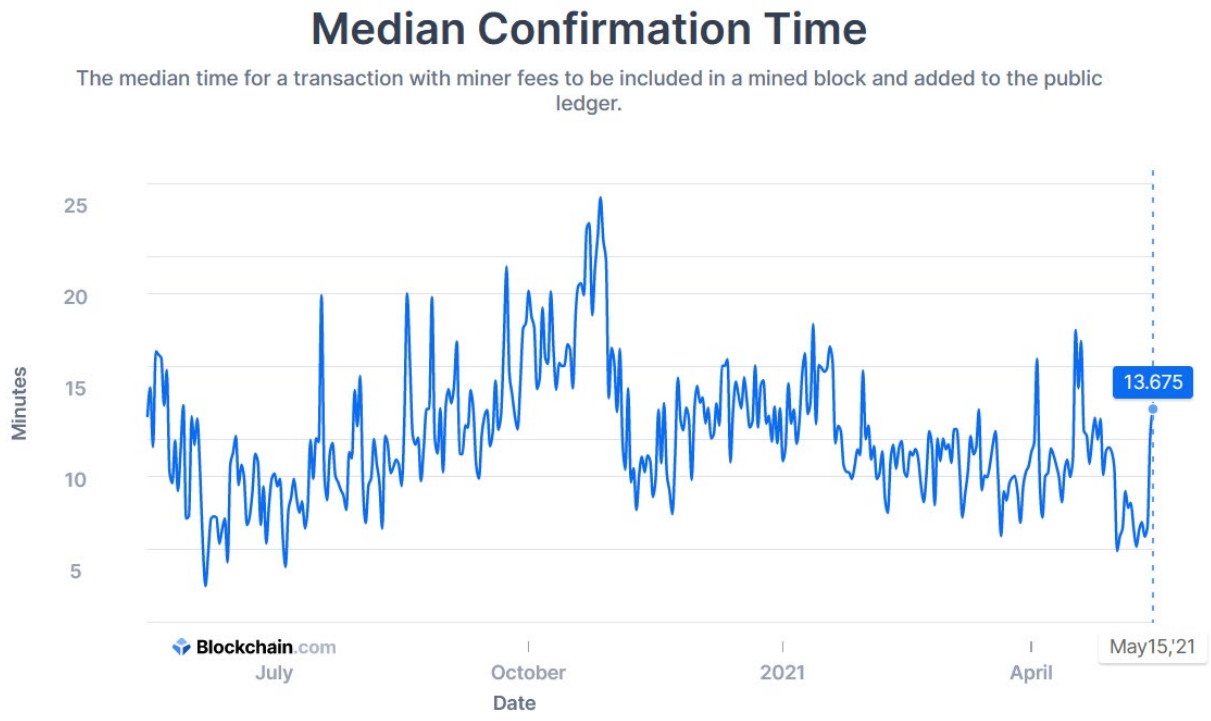


Рис. 3.1.1

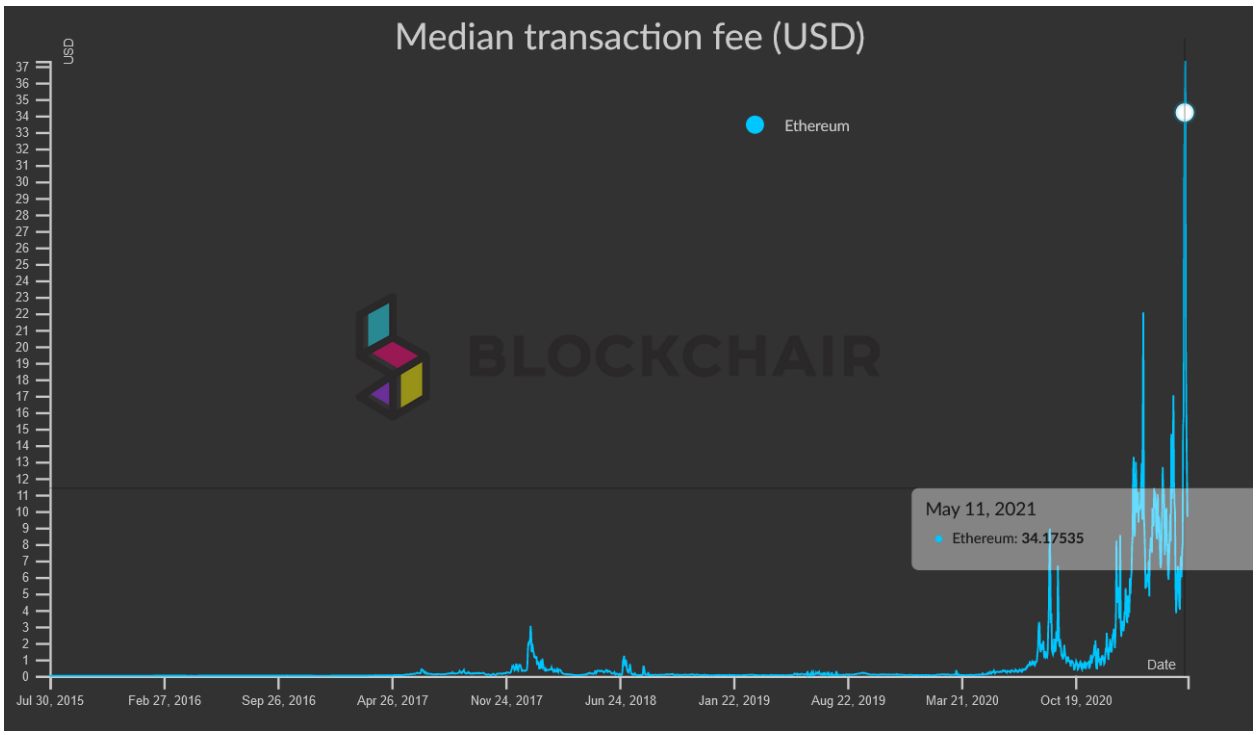


Рис. 3.1.2

На даний момент велика плата за транзакції та їх швидкість наштовхує на думку, що не виправдано реалізовувати всю логіку на блокчейні, треба виносити туди тільки те, що справді необхідно. Очевидно, що чим більше логіки реалізовано на блокчейні, а не на сервері, тим більш децентралізований та надійний застосунок, а значить меншу роль відіграє фактор довіри в успішності проведення операцій між користувачами. Чим більше реалізовано на сервері – тим швидшим та дешевшим у використанні буде застосунок. Але на практиці треба брати до уваги, що звичайні користувачі не будуть звертати увагу на те, що під капотом, їм треба швидкий, надійний та простий у використанні застосунок, отже задача розробника знайти найбільш збалансований та гнучкий варіант під конкретний випадок.

А) Використання «важкого» блокчейну

Підхід, при якому вся логіка застосунку, його сховище перенесені на блокчейн. За такого підходу сервер як ланку можна за бажанням прибрати.

Плюси такого підходу – повна децентралізація, адже клієнти взаємодіють між собою виключно за допомогою блокчейну без використання серверу.

Мінуси описані вище.

Б) Використання «тонкого блокчейну»

Гнучкий підхід з явним використанням серверу. Розробник сам вирішує, як багато він перенесе на сервер. Наприклад, у соціальній мережі нема необхідності зберігати коментарі під фотографіями у блокчейні, куди важливіше зберегти там логіни та паролі. Це рішення призвело до збільшення швидкості без втрати надійності застосунку.

3.2 Варіанти вибору технології

Використання існуючого блокчейну знижує поріг входження розробників у створення децентралізованих застосунків. Такі системи, як Ethereum, Tezos мають можливість створення розумних контрактів на своїй базі. Але за використання чужої платформи для свого застосунку буде стягуватися комісія, тобто із зростанням користувачів вашого застосунку зростатимуть і операційні витрати.

Варіант створення свого блокчейну з нуля, звісно, складніший на моменті розробки. Необхідно повністю створити свою мережу, з криптуванням, перевіркою транзакцій, підтримкою нод тощо. Зате у власному блокчейні не треба платити комісії за транзакції.

Розділ 4. Реалізація платформи для баг-баунті

4.1 Обрані методи розробки

З вищеописаних підходів я вибрав для блокчейн-частини готовий блокчейн Tezos, оскільки для початку розуміння роботи смарт-контрактів та інтеграції блокчейна розробляти власний блокчейн затратно по часу. У Tezos

досить багата та зрозуміла документація, спільнота а також інформативні навчальні курси.

Express – фреймворк для Node JS для створення серверної частини. Node JS – оточення для виконання JavaScript.

Taquito – бібліотека для взаємодії зі смарт-контрактами та блокчейном в цілому. Приклад використання у Додатку C та D.

TezBridge – «гаманець» для підписання транзакцій на стороні клієнта.

SmartPy – високорівнева мова програмування контрактів, яка потім інтрепретується у рідну для Tezos низькорівневу мову Michelson.

SmartPyI IDE – оточення виконання коду, яке надає зручні інструменти розробки та тестування розумних контрактів.

В якості мережі Tezos для розгортання застосунку була використана тестова мережа Edonet. Тестові токени можна отримати за допомогою сайту https://faucet.tzalpha.net/, який генерує акаунти з тестовими токенами tz у тестових мережах, у т.ч. Edonet для функціонування економіки застосунку.

4.2. Роль смарт-контракту

Ключове завдання при створенні баг-баунті платформи на блокчейні - це забезпечення розробника гарантовано вірним звітом по його ресурсу а тестувальника його винагородою. Зрозуміло, що смарт-контракт без штучного інтелекту сам не зможе виконати умову «якщо звіт вірний, то виплатити кошти тестувальнику, а звіт віддати розробнику», тому що невідомо, на яких засадах звіт вважатиметься вірним. Отже, необхідно вводити третю сторону, так званого «суддю». Маємо три сторони в контракті:

- Розробник. Має право покласти кошти на рахунок та подивитися стан сховища контракту. Вводить початкове сховище для контракту і відправляє на сервер для його ініціації. Розробник вкрай зацікавлений, що ввести ті дані (нагорода судді, адресу судді, баланс контракту), про які йшла домовленість із сторонами, оскільки інакше ніхто не буде виконувати свою частину контракту.
- Тестувальник так само може подивитися стан сховища контракту а також переконатися, що зазначена сума лежить на контракті і може безпечно приступати над тестуванням ресурсу. Також може перевірити, чи дійсно у контракті вказаний той суддя, про який була домовленість з розробником.
- Суддя має право присудити кошти розробнику або тестувальнику. Судді за виконання роботи так само буде виплачена сума, узгоджена контрактом. Якщо суддя бачить, що звіт влаштовує вимогам, зазначеним у замовленні, від знімає кошти з контракту на адресу тестувальника, вказану у контракті. Якщо суддя бачить, що робота не зроблена, гроші повертаються назад розробнику. Суддя може перевірити у будь-який момент, чи така винагорода вказана у контракті, про яку йшла домовленість.

Розробник і тестувальник приходять згоди щодо судді, значить вони обоє йому довіряють і довіряють його подальшому рішенню. Вони можуть вибирати за рейтингом або за знайомством. Суддя оголошує свою нагороду. Зрозуміло, що чим більший рейтинг у судді, тим більшим попитом він буде користуватися, тим більше ціну він може поставити.

Смарт-контракт у даному застосунку використовується як рахунок для безпечного зберігання коштів з можливістю їх зняти за настанням певних умов. При створенні цього смарт-контракту сервером у ньому прописується адреса судді, розробника і тестувальника, власники яких будуть мати

виняткові права на дії в цьому контракті. Реалізація коду та тестів контракту наведено у Додатку А та В.

4.3. Загальний огляд структури

Основною задачею було проектування взаємодії між блокчейном, сервером та клієнтом, що підходить для роботи баг-баунті платформи. З огляду на описані вище стратегії було обрано використовувати «тонкий блокчейн», а саме:

- Є сервер, на якому зберігається вся база даних (замовлення, користувачі)
- Розміщення, подання заявок на завдання, обрання судді відбувається не в блокчейні.
- Коли розробник вибрав тестувальника і вони узгодили суддю, то відбувається запит на створення контракту «угоди» між ними на сервер. Було вирішено довірити ініціацію контракту серверу з огляду на забезпечення незмінності коду логіки контракту та щоб позбавити вникання користувача у тонкощі блокчейну і смарт-контрактів.
- Сервер ініціює контракт з відповідною логікою та сховищем і відсилає усім учасникам контракту його адресу в блокчейні.
- Учасники контракту за допомогою цього номеру і вказаного при ініціації контракту власної адреси взаємодіють с контрактом через TezBridge (гаманець, аналог Metamask в Tezos).

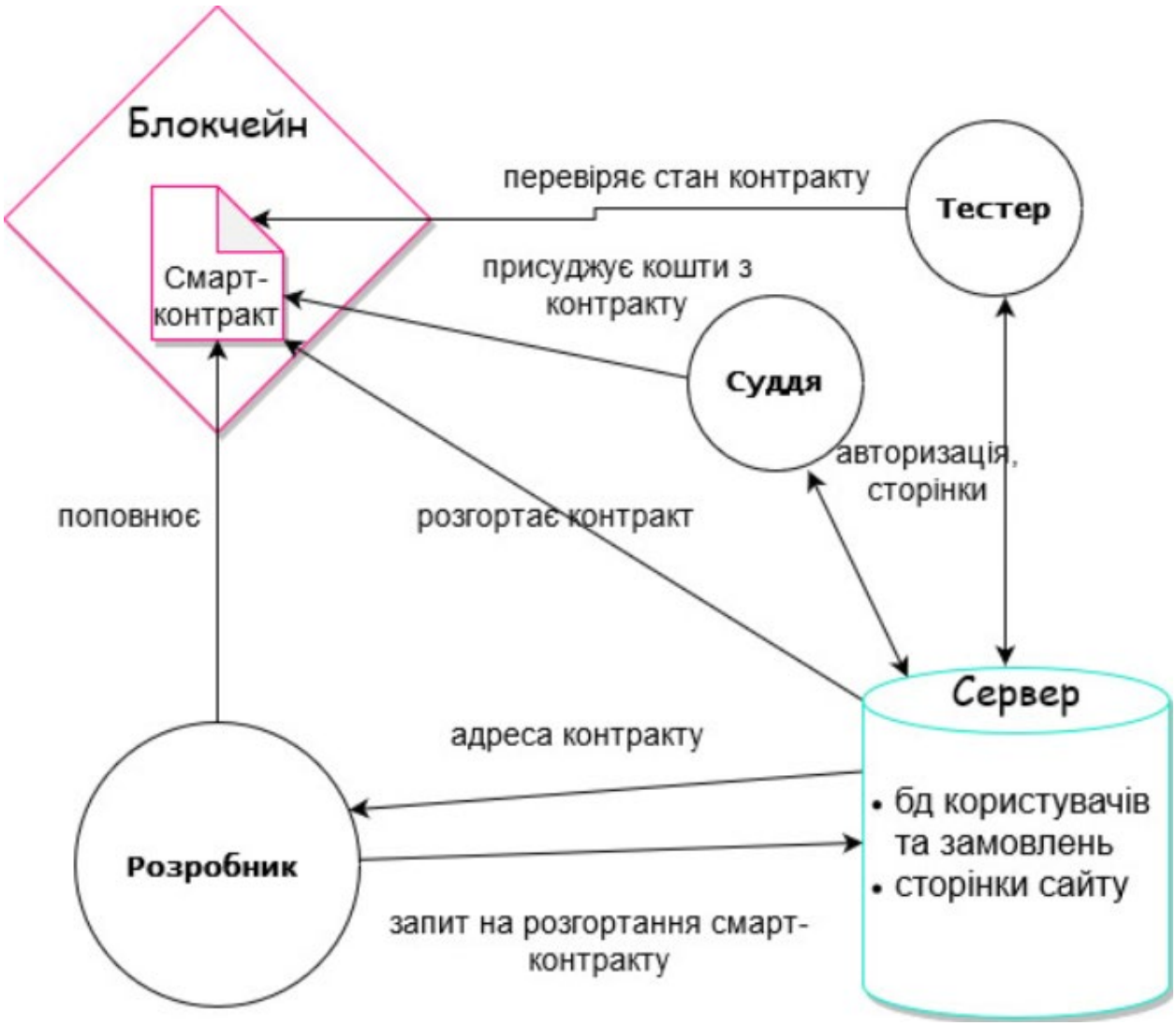


Рис.4.3.1 Комунікація між компонентами

Висновок

Під час написання курсової роботи були досліджені такі поняття, як блокчейн, смарт-контракти, електронно-цифровий підпис а також аспекти побудови практичних застосунків на блокчейні. Була розроблена демонстраційна версія застосунку для баг-баунті та спосіб інтеграції блокчейну в неї. Для цього був розроблений смарт-контракт, який створюється під кожне завдання і допомагає безпечно для кожної із сторін розпоряджатися коштами. Отже, мету досліджень досягнуто.

Перелік використаних джерел

1. Wikipedia [Електронний ресурс]. – URL: https://en.wikipedia.org/wiki/
2. Hackernoon [Електронний ресурс]. – URL: https://hackernoon.com/
3. Umbrella IT [Електронний ресурс]. – URL: https://umbrellait.com/
4. Хабр [Електронний ресурс]. – URL: https://habr.com/
5. TezBridge Docs [Електронний ресурс]. – URL: https://docs.tezbridge.com/
6. SmartPy.io Reference Manual [Електронний ресурс]. – URL: https://smartpy.io/
7. Blockmatics [Електронний ресурс]. – URL: https://blockmatics.teachable.com/
8. Taquito [Електронний ресурс]. – URL: https://tezostaquito.io/

Додаток А

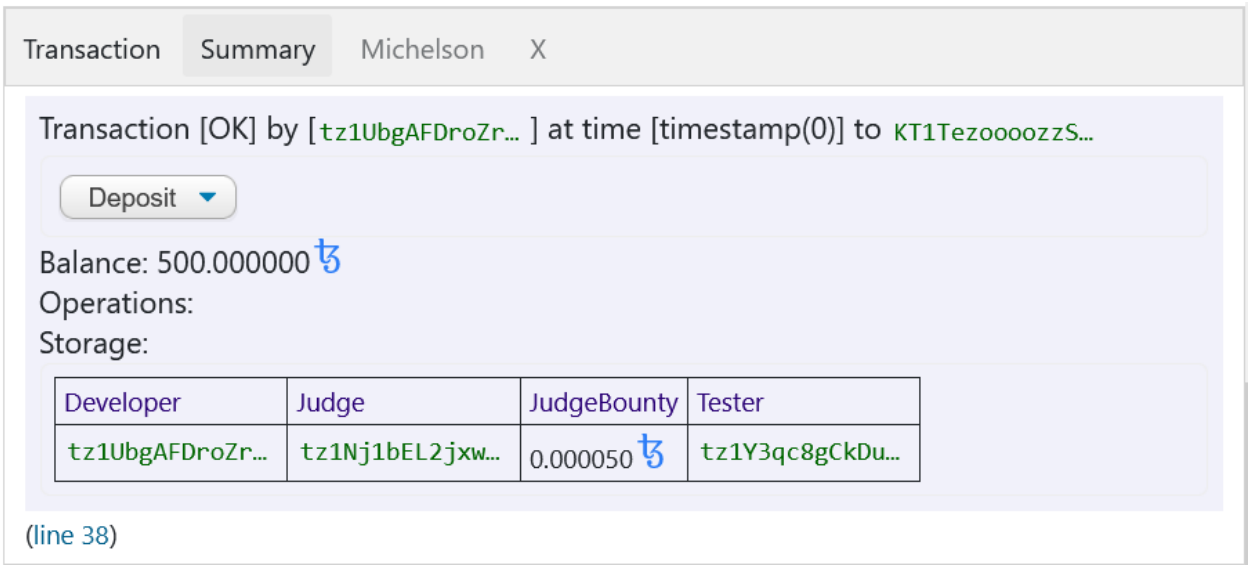
SmartPy код смарт-контракту з тестами

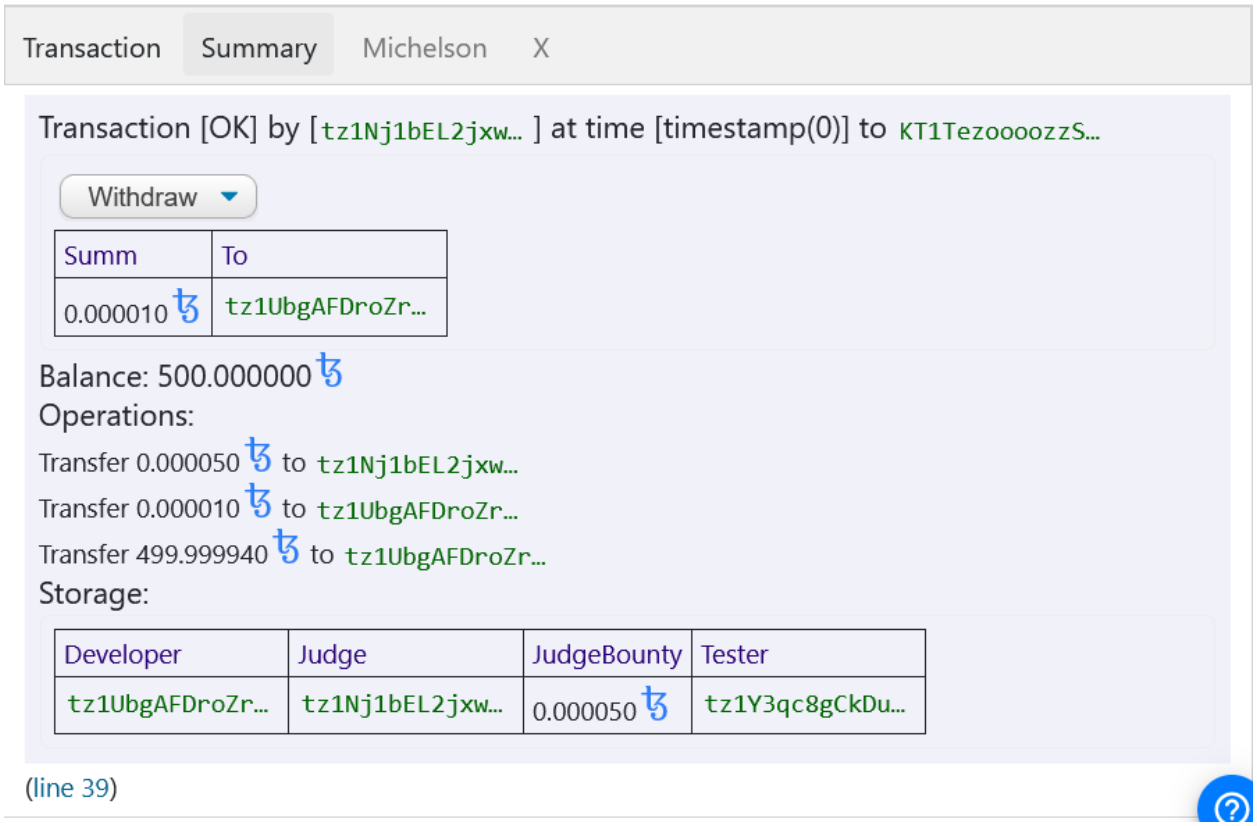
```
import smartpy as sp


class Deal(sp.Contract):
    def __init__(self,developer,tester,judge,judgeBounty):
```

```
        self.init(developer=developer,
                  tester=tester,
                  judge=judge,
                  judgeBounty=judgeBounty)

    @sp.entry_point
    def deposit(self):
        sp.verify(self.data.developer==sp.sender)
        sp.verify(sp.amount>sp.tez(0))

    @sp.entry_point
    def withdraw(self, summ, to):
        sp.verify(self.data.judge==sp.sender)
        sp.if ~ (self.data.developer==to):
            sp.if ~(self.data.tester==to):
                sp.failwith("Incorrect destination address")
        sp.verify(summ+self.data.judgeBounty<=sp.balance)
        sp.send(self.data.judge,self.data.judgeBounty)
        sp.send(to,summ)
        sp.send(self.data.developer,sp.balance-summ-self.data.judgeBounty)

@sp.add_test(name = "Deal")
def test():
    server = sp.test_account("Server")
    dev = sp.test_account("Developer")
    jud = sp.test_account("Judge")
```

```
tes = sp.test_account("Tester")
c1 = Deal(dev.address,tes.address,jud.address,sp.mutez(50))
scenario = sp.test_scenario()
scenario.h1("Deal")
scenario+=c1
scenario+=c1.deposit().run(sender=dev,amount = sp.tez(500))
```

```
scenario+=c1.withdraw(summ=sp.mutez(10),to=dev.address).run(sender=jud)
```





Додаток В

Сгенерований код Michelson

parameter (or (unit %deposit) (pair %withdraw (mutez %summ) (address %to)));

storage (pair (pair (address %developer) (address %judge)) (pair (mutez %judgeBounty) (address %tester)));

code
{
UNPAIR; # @parameter : @storage
IF_LEFT
{
DROP; # @storage
== deposit ==
sp.verify(self.data.developer == sp.sender) # @storage
DUP; # @storage : @storage
CAR; # pair (address %developer) (address %judge) : @storage
CAR; # address : @storage
SENDER; # @sender : address : @storage

```
      COMPARE;          #          int          :          @storage
      EQ;               #          bool         :          @storage
      IF
        {}
        {
          PUSH string "WrongCondition: self.data.developer == sp.sender"; # string :
@storage
          FAILWITH;                  #                  FAILED
        };                           #                  @storage
      #       sp.verify(sp.amount       >       sp.tez(0))       #       @storage
      PUSH          mutez          0;          #          mutez          :          @storage
      AMOUNT;       #       @amount       :       mutez       :       @storage
      COMPARE;          #          int          :          @storage
      GT;               #          bool         :          @storage
      IF
        {}
        {
          PUSH string "WrongCondition: sp.amount > sp.tez(0)"; # string : @storage
          FAILWITH;                  #                  FAILED
        };                           #                  @storage
      NIL          operation;          #          list          operation          :          @storage
    }
    {
      #                  ==                  withdraw                  ==
      # sp.verify(self.data.judge == sp.sender) # @parameter%withdraw : @storage
      SENDER;       #       @sender       :       @parameter%withdraw       :       @storage
      DUP 3;     # @storage : @sender : @parameter%withdraw : @storage
      CAR;       # pair (address %developer) (address %judge) : @sender :
@parameter%withdraw                  :                  @storage
      CDR;       # address : @sender : @parameter%withdraw : @storage
      COMPARE;       #       int       :       @parameter%withdraw       :       @storage
      EQ;            #          bool          :          @parameter%withdraw          :          @storage
      IF
        {}
        {
          PUSH string "WrongCondition: self.data.judge == sp.sender"; # string :
@parameter%withdraw                  :                  @storage
          FAILWITH;                  #                  FAILED
        };           #           @parameter%withdraw           :           @storage
      # if ~ (self.data.developer == params.to): # @parameter%withdraw : @storage
      DUP;       # @parameter%withdraw : @parameter%withdraw : @storage
      CDR;       #       address       :       @parameter%withdraw       :       @storage
      DUP 3;     # @storage : address : @parameter%withdraw : @storage
      CAR;       # pair (address %developer) (address %judge) : address :
@parameter%withdraw                  :                  @storage
```

```
    CAR;      # address : address : @parameter%withdraw : @storage
    COMPARE;  # int : @parameter%withdraw : @storage
    EQ;       # bool : @parameter%withdraw : @storage
    IF
      {}
      {
       # if ~ (self.data.tester == params.to): # @parameter%withdraw : @storage
       DUP;      # @parameter%withdraw : @parameter%withdraw : @storage
       CDR;      # address : @parameter%withdraw : @storage
       DUP 3;    # @storage : address : @parameter%withdraw : @storage
       GET 4;    # address : address : @parameter%withdraw : @storage
       COMPARE;  # int : @parameter%withdraw : @storage
       EQ;       # bool : @parameter%withdraw : @storage
       IF
         {}
         {
          PUSH string "Incorrect destination address"; # string : @parameter%withdraw : @storage
          FAILWITH;                                    # FAILED
         };         # @parameter%withdraw : @storage
      };            # @parameter%withdraw : @storage
    # sp.verify((params.summ + self.data.judgeBounty) <= sp.balance) # @parameter%withdraw : @storage
    BALANCE;  # @balance : @parameter%withdraw : @storage
    DUP 3;    # @storage : @balance : @parameter%withdraw : @storage
    GET 3;    # mutez : @balance : @parameter%withdraw : @storage
    DUP 3;    # @parameter%withdraw : mutez : @balance : @parameter%withdraw : @storage
    CAR;      # mutez : mutez : @balance : @parameter%withdraw : @storage
    ADD;      # mutez : @balance : @parameter%withdraw : @storage
    COMPARE;  # int : @parameter%withdraw : @storage
    LE;       # bool : @parameter%withdraw : @storage
    IF
      {}
      {
       PUSH string "WrongCondition: (params.summ + self.data.judgeBounty) <= sp.balance"; # string : @parameter%withdraw : @storage
       FAILWITH;                                                                          # FAILED
      };        # @parameter%withdraw : @storage
    # sp.send(self.data.judge, self.data.judgeBounty) # @parameter%withdraw : @storage
    NIL operation; # list operation : @parameter%withdraw : @storage
    DUP 3;    # @storage : list operation : @parameter%withdraw : @storage
    CAR;      # pair (address %developer) (address %judge) : list operation : @parameter%withdraw : @storage
```

CDR; # address : list operation : @parameter%withdraw : @storage
CONTRACT unit; # option (contract unit) : list operation : @parameter%withdraw : @storage
IF_SOME
{}
{
PUSH int 22; # int : list operation : @parameter%withdraw : @storage
FAILWITH; # FAILED
}; # @some : list operation : @parameter%withdraw : @storage
DUP 4; # @storage : @some : list operation : @parameter%withdraw : @storage
GET 3; # mutez : @some : list operation : @parameter%withdraw : @storage
UNIT; # unit : mutez : @some : list operation : @parameter%withdraw : @storage
TRANSFER_TOKENS; # operation : list operation : @parameter%withdraw : @storage
CONS; # list operation : @parameter%withdraw : @storage
SWAP; # @parameter%withdraw : list operation : @storage
sp.send(params.to, params.summ) # @parameter%withdraw : list operation : @storage
DUP; # @parameter%withdraw : @parameter%withdraw : list operation : @storage
DUG 2; # @parameter%withdraw : list operation : @parameter%withdraw : @storage
CDR; # address : list operation : @parameter%withdraw : @storage
CONTRACT unit; # option (contract unit) : list operation : @parameter%withdraw : @storage
IF_SOME
{}
{
PUSH int 23; # int : list operation : @parameter%withdraw : @storage
FAILWITH; # FAILED
}; # @some : list operation : @parameter%withdraw : @storage
DUP 3; # @parameter%withdraw : @some : list operation : @parameter%withdraw : @storage
CAR; # mutez : @some : list operation : @parameter%withdraw : @storage
UNIT; # unit : mutez : @some : list operation : @parameter%withdraw : @storage
TRANSFER_TOKENS; # operation : list operation : @parameter%withdraw : @storage
CONS; # list operation : @parameter%withdraw : @storage
sp.send(self.data.developer, (sp.balance - params.summ) - self.data.judgeBounty) # list operation : @parameter%withdraw : @storage

```
    DUP 3;     # @storage : list operation : @parameter%withdraw : @storage
    CAR;       # pair (address %developer) (address %judge) : list operation : @parameter%withdraw : @storage
    CAR;       # address : list operation : @parameter%withdraw : @storage
    CONTRACT unit; # option (contract unit) : list operation : @parameter%withdraw : @storage
    IF_SOME
      {}
      {
       PUSH int 24; # int : list operation : @parameter%withdraw : @storage
       FAILWITH;    # FAILED
      }; # @some : list operation : @parameter%withdraw : @storage
    DUP 4;     # @storage : @some : list operation : @parameter%withdraw : @storage
    GET 3;     # mutez : @some : list operation : @parameter%withdraw : @storage
    DIG 3;     # @parameter%withdraw : mutez : @some : list operation : @storage
    CAR;       # mutez : mutez : @some : list operation : @storage
    BALANCE;   # @balance : mutez : mutez : @some : list operation : @storage
    SUB;       # mutez : mutez : @some : list operation : @storage
    SUB;       # mutez : @some : list operation : @storage
    UNIT;      # unit : mutez : @some : list operation : @storage
    TRANSFER_TOKENS; # operation : list operation : @storage
    CONS;      # list operation : @storage
   };          # list operation : @storage
  NIL operation; # list operation : list operation : @storage
  SWAP;        # list operation : list operation : @storage
  ITER
   {
    CONS;      # list operation : @storage
   };          # list operation : @storage
  PAIR;        # pair (list operation) @storage
 };
```

Додаток С

Код, що пов’язує клієнта та блокчейн

```
Tezos.setProvider({
    rpc: 'https://edonet.smartpy.io',
    signer: new TezBridgeSigner()
});
```

```
async function getBalance() {
    let address = document.getElementById("address_field").value;
    console.log(address);
    const balance = await Tezos.tz.getBalance(address);
    document.getElementById("balance_field").value = balance.toString();
}

async function deposit() {
    let value = document.getElementById("deposit_value").value;
    let address = document.getElementById("address_field").value;
    const contract = await Tezos.contract.at(address);
    const transaction = await contract.methods
        .deposit('')
        .send({amount: value});
    if (transaction.status == "applied") alert("Successful");
    else alert("error");
}
```

Додаток D

Код, що пов’язує сервер і блокчейн

```
const taquito = require('@taquito/taquito');
const signer = require('@taquito/signer');
const config = require('../configs/default.json');
const InMemorySigner = signer.InMemorySigner;
const Tezos = new taquito.TezosToolkit(config.blockchain.rpc);
const dealContract = require('../contracts/Deal.json');

InMemorySigner.fromSecretKey('edskRt8xQkz3scoavUNurWhGpLafvTv1NR2WhwpBPrvwi6v
AHk2Ec7YghM74ubzkkFJPFKqgSE4SVKtL1R9Uwn5RJiXpfBWUEm')
    .then((theSigner) => {
        Tezos.setProvider({signer: theSigner});
        //We can access the public key hash
        return Tezos.signer.publicKeyHash();
    })
    .then((publicKeyHash) => {
        console.log(`The public key hash associated is: ${publicKeyHash}.`);
    })
    .catch((error) => console.log(`Error: ${error} ${JSON.stringify(error,
null, 2)}`));

async function createContract(devAdd, testerAdd, judgeAdd, judgeBounty) {
    let originationOp = await Tezos.contract
        .originate({
            code: dealContract,
            storage: {
                developer: devAdd,
                judge: judgeAdd,
                judgeBounty: judgeBounty,
                tester: testerAdd
            },
        });
    console.log(`Waiting for confirmation of origination for
${originationOp.contractAddress}...`);
    originationOp.contract()
        .then((contract) => {
            console.log(`Origination completed.`);
        })
        .catch((error) => console.log(`Error: ${JSON.stringify(error, null,
```

```
2)}`));
    return originationOp.contractAddress;
};

module.exports = {createContract}
```