Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

Технологія блокчейн та розподілені системи

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 1

Розгортання систем Ethereum та криптовалют

Виконали:

студенти V курсу ФТІ

групи ФБ-31мп

Снігур А. Ю.

Тислицький Д. В.

Чорний А. Ю.

Перевірила:

Селюх П. В.

Мета роботи: Отримання навичок налаштування платформ виконання смарт-контрактів та криптовалют
Завдання: Провести налаштування обраної системи та виконати тестові операції в системі ethereum

Теоретична частина:

В Ethereum стан складається з об'єктів, які називаються "accounts", причому кожен account має 20-байтну адресу, а переходи стану - це прямі перекази вартості та інформації між обліковими записами. Акаунт Ethereum містить чотири поля:

- Nonce, лічильник, який використовується для того, щоб переконатися, що кожна транзакція може бути оброблена тільки один раз
- поточний баланс ефіру на рахунку
- код контракту акаунта, якщо він присутній
- Сховище акаунта (за замовчуванням порожнє)

Термін "транзакція" використовується в Ethereum для позначення підписаного пакету даних, який зберігає повідомлення, що надсилається з зовнішнього облікового запису. Транзакції містять:

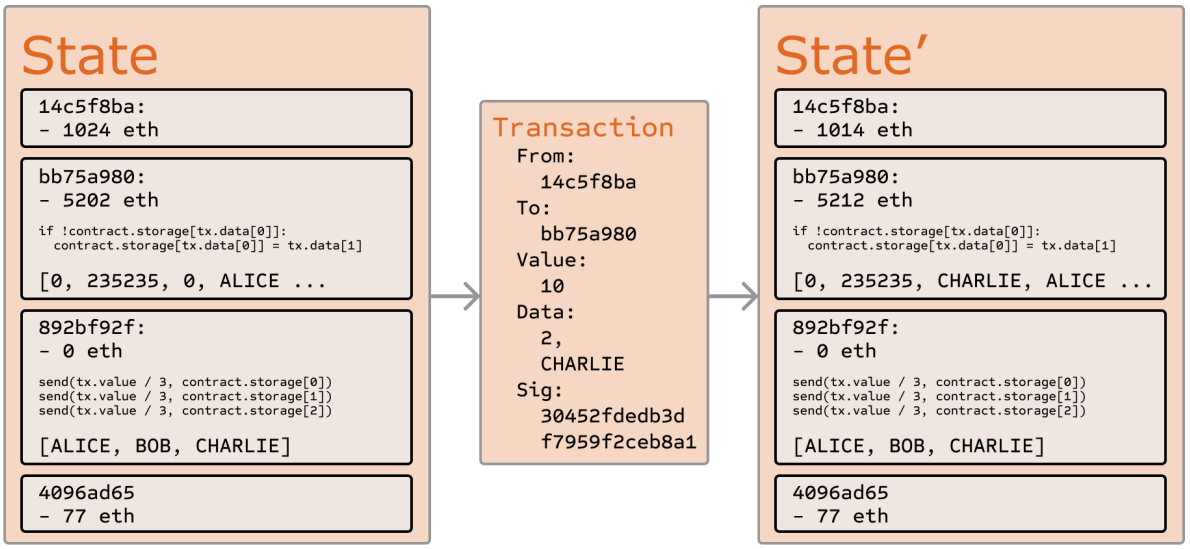
- одержувача повідомлення
- підпис, що ідентифікує відправника
- Кількість ефіру для переказу від відправника до одержувача
- необов'язкове поле даних
- Значення STARTGAS, що представляє максимальну кількість обчислювальних кроків, які дозволяється зробити при виконанні транзакції
- Значення GASPRICE, що представляє плату, яку відправник сплачує за кожен обчислювальний крок

Контракти мають можливість надсилати "повідомлення" іншим контрактам. Повідомлення - це віртуальні об'єкти, які ніколи не серіалізуються і існують тільки в середовищі виконання Ethereum. Повідомлення містить:

- Відправник повідомлення (неявний)
- одержувача повідомлення
- Кількість ефіру для передачі разом з повідомленням

- необов'язкове поле даних
- Значення STARTGAS

Ethereum State Transition Function



Функцію переходу стану Ethereum, APPLY(S,TX) -> S', можна визначити наступним чином:

- Перевірка, чи транзакція добре сформована (тобто має правильну кількість значень), підпис дійсний, а nonce збігається з nonce в обліковому записі відправника. Якщо ні, повернути помилку.
- Обчислити комісію за транзакцію як STARTGAS * GASPRICE і визначити адресу відправника за підписом. Відняти комісію від балансу рахунку відправника і збільшити nonce відправника. Якщо на рахунку недостатньо коштів, повернути помилку.
- Ініціалізувати GAS = STARTGAS і зняти певну кількість газу за байт, щоб оплатити байти в транзакції.
- Перевести вартість транзакції з рахунку відправника на рахунок отримувача. Якщо рахунок одержувача ще не існує, створити його. Якщо обліковий запис одержувача є контрактом, виконати код контракту до завершення виконання.
- Якщо передача вартості не відбулася через те, що у відправника не вистачило грошей або виконання коду закінчилося, відмінити всі зміни стану, крім сплати комісійних, і додати комісійні на рахунок майнера.
- В іншому випадку, повернути комісію за весь газ, що залишився, відправнику, а комісію, сплачену за спожитий газ, надіслати майнеру.

Код в контрактах Ethereum написаний низькорівневою мовою байт-коду на основі стеку, яка називається "код віртуальної машини Ethereum" або "код EVM". Код складається з серії байт, де кожен байт представляє операцію. Загалом, виконання коду являє собою нескінченний цикл, який складається з повторного виконання операції при поточному значенні програмного лічильника (який починається з нуля) і подальшого збільшення програмного лічильника на одиницю, поки не буде досягнутий кінець коду або не буде виявлена помилка чи інструкція STOP або RETURN. Операції мають доступ до трьох типів простору для зберігання даних:

- Стек, контейнер, що працює за принципом "останнім прийшов - першим вийшов", до якого можна заносити і виносити значення
- Пам'ять, нескінченно розширюваний байтовий масив
- Довгострокове сховище контракту, сховище ключів/значень. На відміну від стеку та пам'яті, які скидаються після завершення обчислень, сховище зберігається протягом тривалого часу.

Код також може отримати доступ до значення, відправника і даних вхідного повідомлення, а також до даних заголовка блоку, і код також може повернути байтовий масив даних на виході.

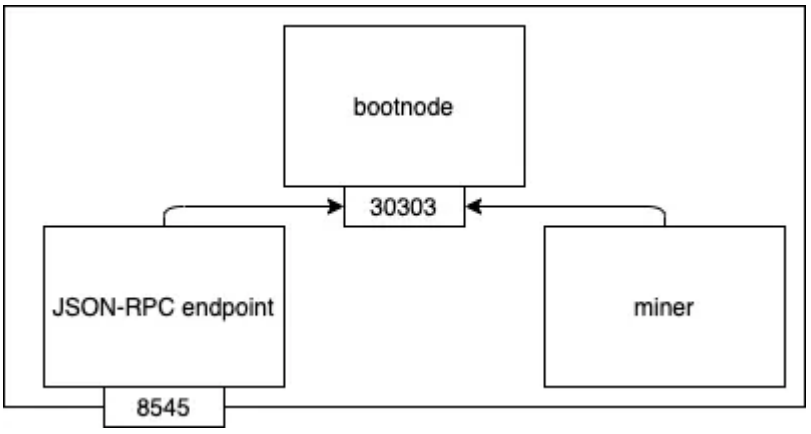
У контексті мережі Ethereum термін "gas" використовується для вимірювання обчислювальних ресурсів, необхідних для виконання операцій або конкретних функцій на мережі. Одна з основних ідеї Ethereum полягає в тому, що кожна операція (наприклад, виконання смарт-контракту або переказ ефірів) має свою вартість у газі, яка визначається складністю та ресурсоємністю операції.
Основні аспекти пов'язані з gas включають:

- Вартість операції у газі: Кожна операція в мережі Ethereum має вартість у газі. Ця вартість визначається різними факторами, такими як обсяг обчислень, необхідний для виконання операції, та складність самої операції.
- Ціна газу: Це вартість одиниці газу в ефірі. Вона визначається ринковим попитом і пропозицією та може змінюватися з часом.
- Вартість операції: Вартість операції в ефірі розраховується як добуток вартості операції у газі на ціну газу.

- Ліміт газу: Кожен блок в мережі Ethereum має обмеження на загальну кількість газу, яку можна витратити на виконання операцій.

Загально, використання концепції газу дозволяє забезпечити економічну стабільність мережі, оскільки вартість операцій залежить від їхньої складності та ресурсоємності, що визначається ринковим попитом та пропозицією

1. **Bootnode** – the bootstrap node that is used for peer discovery purpose. It listens on port 30303, the other nodes joining the network connect to this bootnode first.
2. **JSON-RPC endpoint** – this node exposes JSON-RPC API over HTTP endpoint on port 8545. We will publish the port 8545 of this node container to the host machine to allow external interaction with this private blockchain.
3. **Miner** – this node is responsible for mining (the process of creating a new block in our blockchain). When the miner node successfully mines a new block, it receives the rewards into the configured account.



Genesis block

Щоб створити приватний блокчейн Ethereum, клієнту Ethereum потрібна деяка інформація для створення першого блоку, який називається genesis

```
{
  "config": {
    "chainId": 1214,
    "homesteadBlock": 0,
```

```
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "ethash": {}
  },
  "difficulty": "1",
  "gasLimit": "12000000",
  "alloc": {}
}
```

- **chainId** - це ідентифікатор, який повідомляє вузлам, в якому блокчейні вони знаходяться. Ідентифікатор chainId був введений в EIP-155 для захисту від повторного відтворення. Ми встановили його на 1214, щоб уникнути конфлікту з публічними мережами Ethereum.
- **homesteadBlock** 0 - означає, що використовується реліз платформи Ethereum Homestead. Homestead є другою основною версією платформи Ethereum і є першим production випуском платформи Ethereum
- **eip150Block**: Цей параметр встановлює номер блоку, з якого починають діяти зміни, внесені EIP-150. EIP-150 впроваджує деякі зміни у механізм валідації підпису для забезпечення захисту від атак типу розширення ряду інструкцій (или "gas limit manipulation attack") та зменшенням оплати газу для операцій SLOAD.
- **eip155Block**: Цей параметр встановлює номер блоку, з якого вступає в дію EIP-155. EIP-155 стосується вдосконалення протоколу Ethereum у сфері підпису транзакцій, що допомагає у запобіганні використанню транзакцій для атак типу повторний підпис.
- **eip158Block**: Цей параметр встановлює номер блоку, з якого вступає в дію EIP-158. EIP-158 впроваджує зміни щодо обробки операцій з пам'яттю, зокрема відсуває оплату газу за додатковий використаний простір пам'яті, що стимулює більш ефективне використання пам'яті
- **difficulty** - Скалярне значення, що відповідає рівню складності, застосованому під час виявлення цього блоку. Воно визначає мету видобутку, яку можна обчислити на основі рівня складності попереднього блоку і мітки часу. Чим вищий рівень складності, тим статистично більше обчислень повинен виконати майнер, щоб знайти дійсний блок.

- **gasLimit** - Скалярне значення, що дорівнює поточному ліміту витрат газу для всього ланцюжка на блок. У нашому випадку високе, щоб уникнути обмеження цим порогом під час тестів. Примітка: це не означає, що ми не повинні звертати увагу на споживання газу в наших контрактах
- **alloc** - Дозволяє визначити список попередньо заповнених гаманців. Це специфічна для Ethereum функціональність для управління періодом "попереднього продажу Ефіру". Оскільки ми можемо добувати локальний Ефір швидко, ми не використовуємо цю опцію.
- **ethash -** це означає, що наш блокчейн буде використовувати доказ роботи в якості механізму консенсусу

Dockerfile

```
FROM ethereum/client-go:v1.10.15
ARG ACCOUNT_PASSWORD
COPY genesis.json /tmp
RUN geth init /tmp/genesis.json \
   && rm -f ~/.ethereum/geth/nodekey \
   && echo ${ACCOUNT_PASSWORD} > /tmp/password \
   && geth account new --password /tmp/password \
   && rm -f /tmp/password

ENTRYPOINT ["geth"]
```

Ми використовуємо офіційний образ Geth з Docker Hub як базовий образ. Використовуємо genesis.json для ініціалізації блоку genesis.Файл nodekey створюється під час ініціалізації Geth, він створюється в папці під назвою geth у каталозі даних, який (за замовчуванням) розташований у ~/.ethereum. Файл nodekey використовується для створення Geth enode, який є свого роду ідентифікатором кожного вузла в мережі Ethereum. Оскільки ми будемо використовувати той самий образ, створений із Dockerfile, для запуску всіх вузлів у нашій мережі Ethereum, нам потрібно видалити файл nodekey, створений під час ініціалізації Geth. Таким чином, файл буде відтворено з іншим ключем під час запуску кожного вузла, інакше кожен вузол у нашій приватній мережі матиме той самий enode, що не дозволить вузлам з’єднуватися та синхронізуватися разом. Створюємо новий обліковий запис для нашого блокчейну Ethereum і видаляємо файл пароля

Docker compose file

Bootnode

```
 geth-bootnode:
```

```
  hostname: geth-bootnode
  env_file:
    - .env
  image: geth-client
  build:
    context: .
    args:
      - ACCOUNT_PASSWORD=${ACCOUNT_PASSWORD}
  command:

--nodekeyhex="b0ac22adcad37213c7c565810a50f1772291e7b0ce53fb73e7ec2a3c75bc13b5"
    --nodiscover
    --ipcdisable
    --networkid=${NETWORK_ID}
    --netrestrict="172.16.254.0/28"
  networks:
    priv-eth-net
```

nodekeyhex – ми вказуємо ключ вузла для вузла початкового завантаження, щоб попередньо визначити enode (оскільки вона генерується з ключа вузла) цього вузла початкового завантаження, щоб використовувати його для налаштування інших вузлів.
nodiscover – цьому вузлу не потрібно виявляти інші вузли, оскільки інші підключатимуться до нього під час приєднання до мережі.
ipcdisable – щоб зробити вузол більш легким, оскільки він використовується лише як вузол початкового завантаження.
networkid – вкажіть ідентифікатор мережі, кожен вузол в одній мережі Ethereum повинен мати однаковий networkid. Уникайте значень, які конфліктують із публічною мережею Ethereum.
netrestrict – приймати підключення лише від вузлів у межах діапазону CIDR

Endpoint

```
 geth-rpc-endpoint:
  hostname: geth-rpc-endpoint
  env_file:
    - .env
  image: geth-client
  depends_on:
    - geth-bootnode
  command:

--bootnodes="enode://af22c29c316ad069cf48a09a4ad5cf04a251b411e45098888d114c6dd7f489
a13786620d5953738762afa13711d4ffb3b19aa5de772d8af72f851f7e9c5b164a@geth-bootnode:30
303"
    --allow-insecure-unlock
    --http
    --http.addr="0.0.0.0"
    --http.api="eth,web3,net,admin,personal"
    --http.corsdomain="*"
    --networkid=${NETWORK_ID}
    --netrestrict="172.16.254.0/28"
  ports:
    - "8545:8545"
  networks:
    priv-eth-net
```

bootnodes – вкажіть список вузлів для підключення для виявлення одноранговых вузлів
allow-insecure-unlock – щоб дозволити розблокувати обліковий запис, який ми створили через HTTP. Це небезпечно, якщо кінцева точка піддається зовнішньому впливу, подумайте про те, щоб видалити цей параметр у робочому середовищі.
http – щоб увімкнути JSON-RPC через протокол HTTP.
http.api – список API для ввімкнення через інтерфейс HTTP-RPC. У робочому середовищі вказуйте лише ті, які потрібні.
http.addr – установіть значення 0.0.0.0, щоб приймати HTTP-з'єднання на всіх IP-адресах контейнера вузла.
http.corsdomain – щоб дозволити підключення з веб-сторінок різних джерел.

Miner

```
 geth-miner:
   hostname: geth-miner
   env_file:
     - .env
   image: geth-client
   depends_on:
     - geth-bootnode
   command:

--bootnodes="enode://af22c29c316ad069cf48a09a4ad5cf04a251b411e45098888d114c6dd7f489
a13786620d5953738762afa13711d4ffb3b19aa5de772d8af72f851f7e9c5b164a@geth-bootnode:30
303"
     --mine
     --miner.threads=1
     --networkid=${NETWORK_ID}
     --netrestrict="172.16.254.0/28"
   networks:
     priv-eth-net
```

mine – щоб увімкнути майнінг для цього вузла
miner.threads – вкажіть кількість потоків ЦП для майнінгу
miner.etherbase – вкажіть адресу облікового запису для отримання винагород за майнінг. Ми не вказуємо це тут, тому винагороди надходять до основного облікового запису, який ми створили за умовчанням.

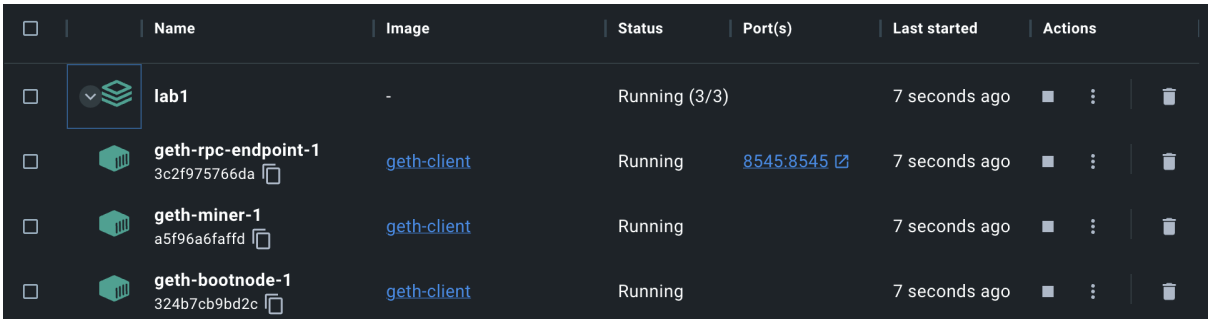
Network configuration

```
priv-eth-net:
  driver: bridge
  ipam:
    config:
    - subnet: 172.16.254.0/28
```

Практична частина

запуск контейнерів

```
kittenrage@kittenrage-222 lab1 % docker-compose up -d
[+] Running 3/3
 ⠿ Container lab1-geth-bootnode-1      Started                                    0.4s
 ⠿ Container lab1-geth-miner-1         Started                                    0.9s
 ⠿ Container lab1-geth-rpc-endpoint-1  Started                                    0.9s
```



Перш за все, перевіримо підключення вузлів

Оскільки ми комунікуємо з вузлом кінцевої точки RPC, ми побачимо, що вузол, до якого він підключається, є bootnode (видно значення enode таке саме, як ми встановили як параметри bootnode у файлі docker-compose.yml).

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 1,
    "method": "admin_peers",
    "params": []
}'
{"jsonrpc":"2.0","id":1,"result":[{"enode":"enode://af22c29c316ad069cf48a09a4ad5cf04a251b411e45098888d114c6dd7f489a13786620d5953738762afa13711d4ffb3b19aa5de772d8af72f851f7e9c5b164a@172.16.254.2:30303","id":"f19e412334ea6bbcbb227e98e43121e27407a0d6cd1d7966ad489c6fd18c4b3d","name":"Geth/v1.10.15-stable-8be800ff/linux-arm64/go1.17.5","caps":["eth/66","snap/1"],"network":{"localAddress":"172.16.254.4:51034","remoteAddress":"172.16.254.2:30303","inbound":false,"trusted":false,"static":false},"protocols":{"eth":{"version":66,"difficulty":88257462,"head":"0x91fda97ccb604386c9d59ea258980ee5d66b9766746bbdc5e595b81862763796"},"snap":{"version":1}}}]}
```

Потім подивимось номер останнього блоку блокчейну

Через деякий час після запуску вузлів ми повинні побачити число більше 0x0, що означає, що наш вузол майнера вже створив інші блоки після блоку genesis

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 2,
    "method": "eth_blockNumber",
    "params": []
}'
{"jsonrpc":"2.0","id":2,"result":"0x260"}
```

Далі ми отримаємо адресу основного облікового запису, створеного під час створення мережі

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 3,
    "method": "eth_accounts",
    "params": []
}'
{"jsonrpc":"2.0","id":3,"result":["0x4738bfba73ac5b25874a9c61ae75a39b294c61b6"]}
```

Перевіряємо баланс на акаунті. Баланс у хексі та в одиницях wei
Якщо останній блок нашого блокчейну більше 0x0, ми повинні побачити ненульовий баланс на рахунку. Оскільки цей перший обліковий запис, який ми створили, він за замовчуванням отримує винагороду за майнінг.

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "eth_getBalance",
    "params": [
        "0x4738bfba73ac5b25874a9c61ae75a39b294c61b6",
        "latest"
    ]
}'
{"jsonrpc":"2.0","id":4,"result":"0x4547d014dda5380000"}
```

Створимо новий аккаунт для тестування функції переказу коштів

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'http://localhost:8545' \
--header 'Content-type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 5,
    "method": "personal_newAccount",
    "params": [
        "d4n1il"
    ]
}'
{"jsonrpc":"2.0","id":5,"result":"0x1b53d15d4358392cfff6b4cdd935670f8b3e77df"}
```

Перед надсиланням коштів, ми розблокуємо аккаунт.

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'http://localhost:8545' \
--header 'Content-type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 6,
    "method": "personal_unlockAccount",
    "params": [
        "0x4738bfba73ac5b25874a9c61ae75a39b294c61b6",
        "5uper53cr3t"
    ]
}'
{"jsonrpc":"2.0","id":6,"result":true}
```

Надсилаємо один етер.

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 7,
    "method": "eth_sendTransaction",
    "params": [
        {
            "from": "0x4738bfba73ac5b25874a9c61ae75a39b294c61b6",
            "to": "0x1b53d15d4358392cfff6b4cdd935670f8b3e77df",
            "value": "0xf4240"
        }
    ]
}'
{"jsonrpc":"2.0","id":7,"result":"0x3a5d11f8e25ecbeb91f1db3b4b13f1f26c4ed0ee888c1c7dede8394657ee9f03"}
```

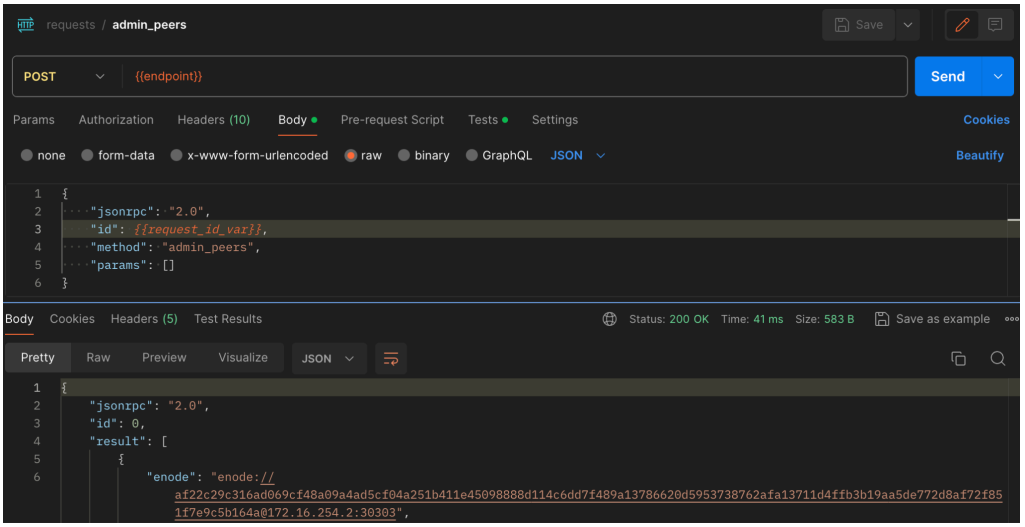
Перевіряємо статус транзакції за її хешем.

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 8,
    "method": "eth_getTransactionByHash",
    "params": ["0x3a5d11f8e25ecbeb91f1db3b4b13f1f26c4ed0ee888c1c7dede8394657ee9f03"]
}'
{"jsonrpc":"2.0","id":8,"result":{"blockHash":"0xc37b6525a22b3321b4029ebfdc9c4ce2fe3f437476308b6caa8843e7
81cbd920","blockNumber":"0x2bc","from":"0x4738bfba73ac5b25874a9c61ae75a39b294c61b6","gas":"0x5208","gasPr
ice":"0x3b9aca00","hash":"0x3a5d11f8e25ecbeb91f1db3b4b13f1f26c4ed0ee888c1c7dede8394657ee9f03","input":"0x
","nonce":"0x0","to":"0x1b53d15d4358392cfff6b4cdd935670f8b3e77df","transactionIndex":"0x0","value":"0xf42
40","type":"0x0","v":"0x99f","r":"0x324cbf0fd0338ea7ad3018836ce75d521265b7a2bf0bf2e1529a65e33d57bd2c","s"
:"0x5875b35ecaf69ed2b2219ca1e7cdc4a5ed3fa14d5d597b40b00184364035216f"}}
```

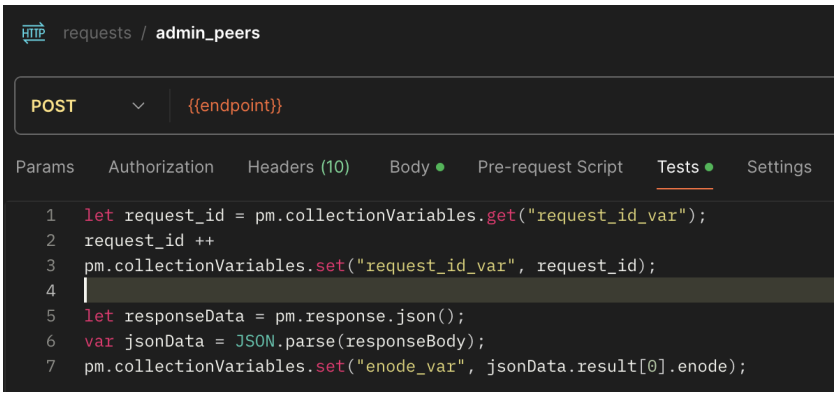
Перевіряємо баланс отримувача.

```
kittenrage@kittenrage-222 lab1 % curl --location --request POST 'localhost:8545' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc": "2.0",
    "id": 9,
    "method": "eth_getBalance",
    "params": [
        "0x1b53d15d4358392cfff6b4cdd935670f8b3e77df",
        "latest"
    ]
}'
{"jsonrpc":"2.0","id":9,"result":"0xf4240"}
```

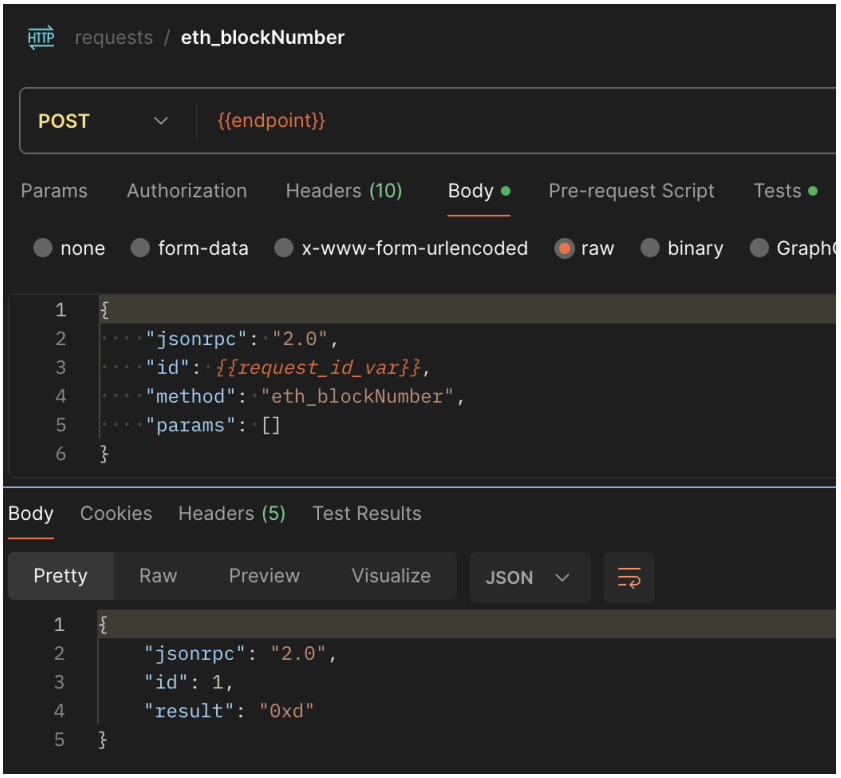
Для зручності було також створено postman колекцію демонстрація запитів у тому ж порядку



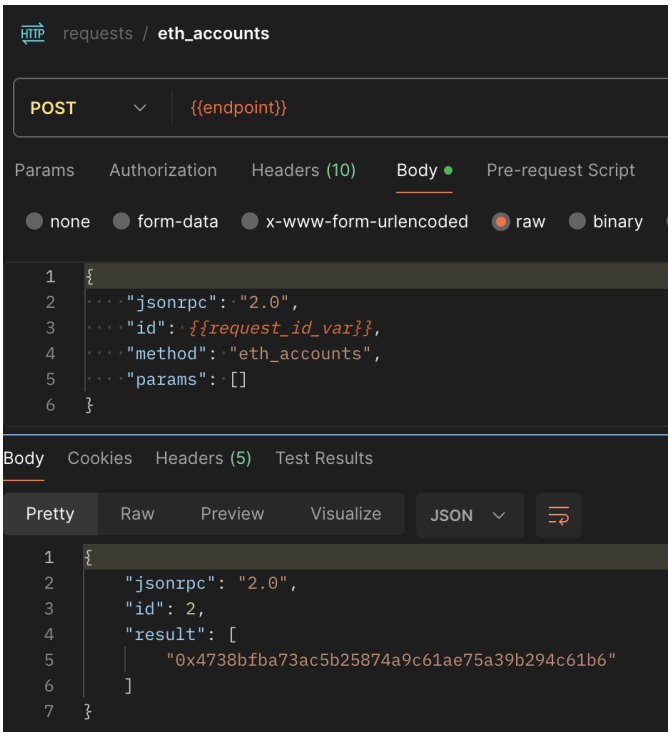
Тест скрипт для оновлення змінних



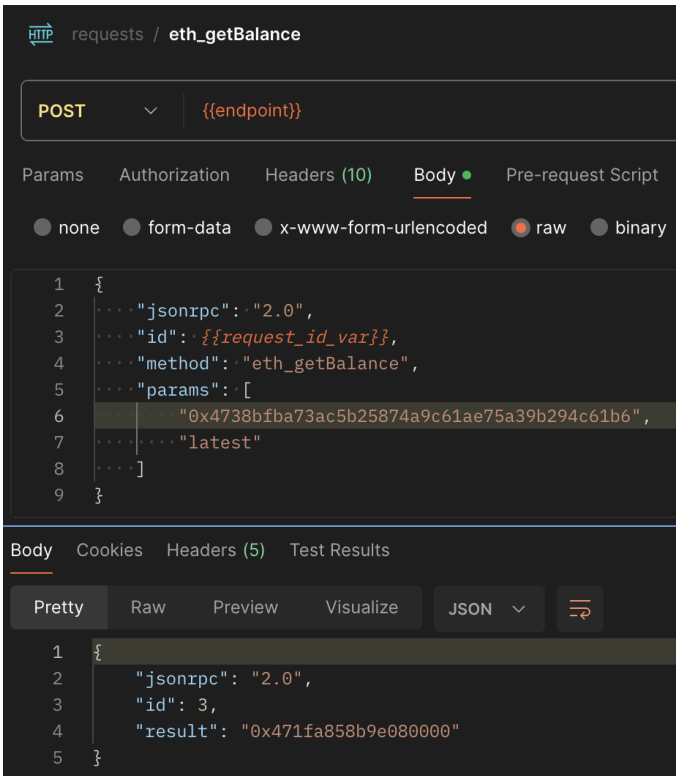
Останній блок в мережі



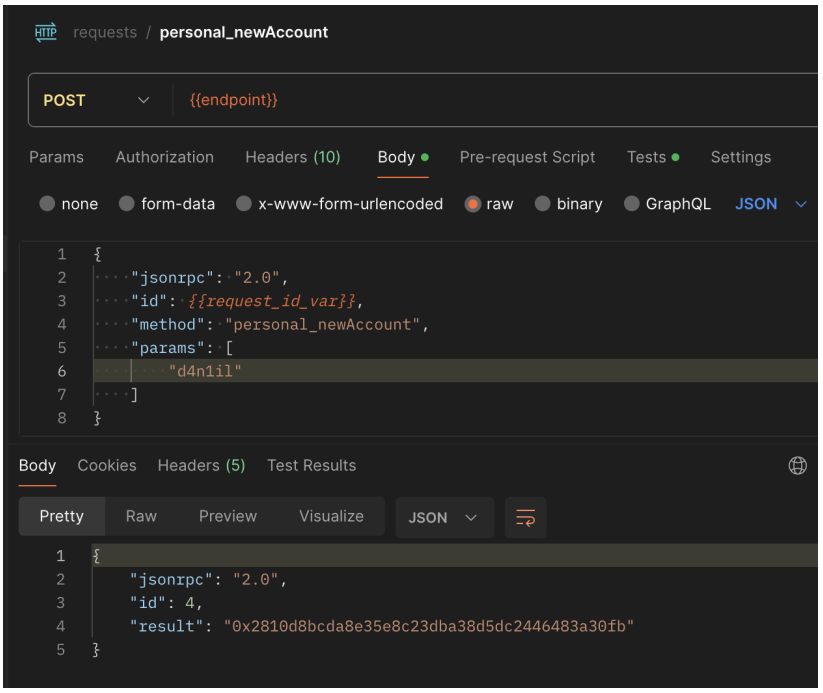
Список всіх акаунтів у мережі



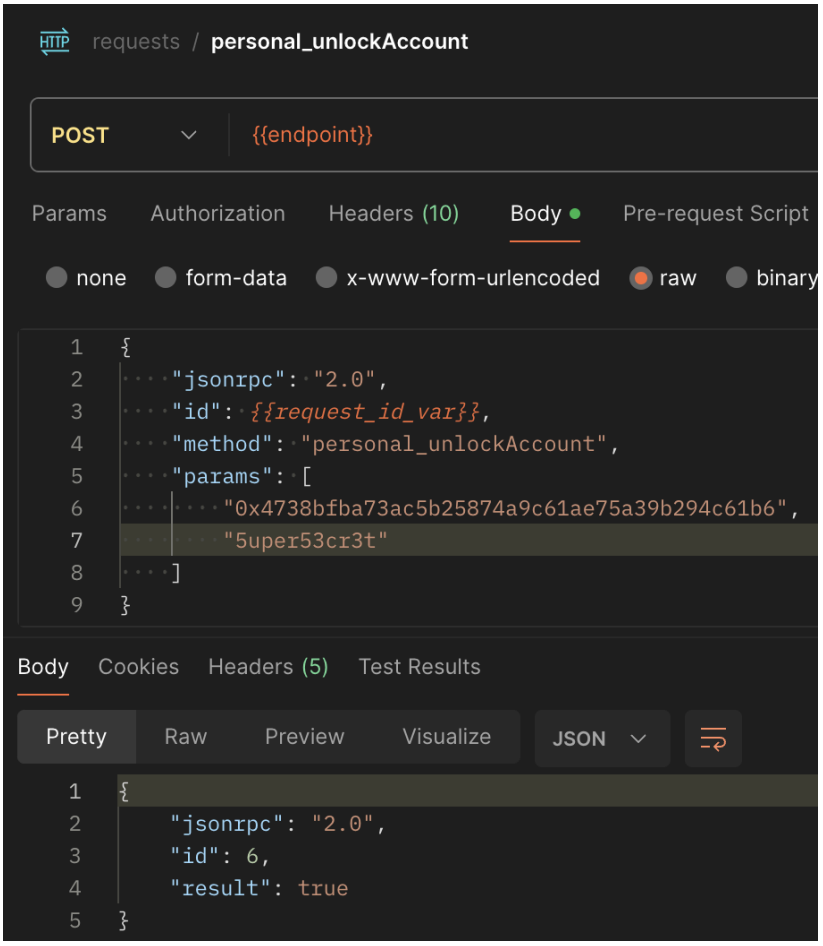
Перевірка балансу на акаунті



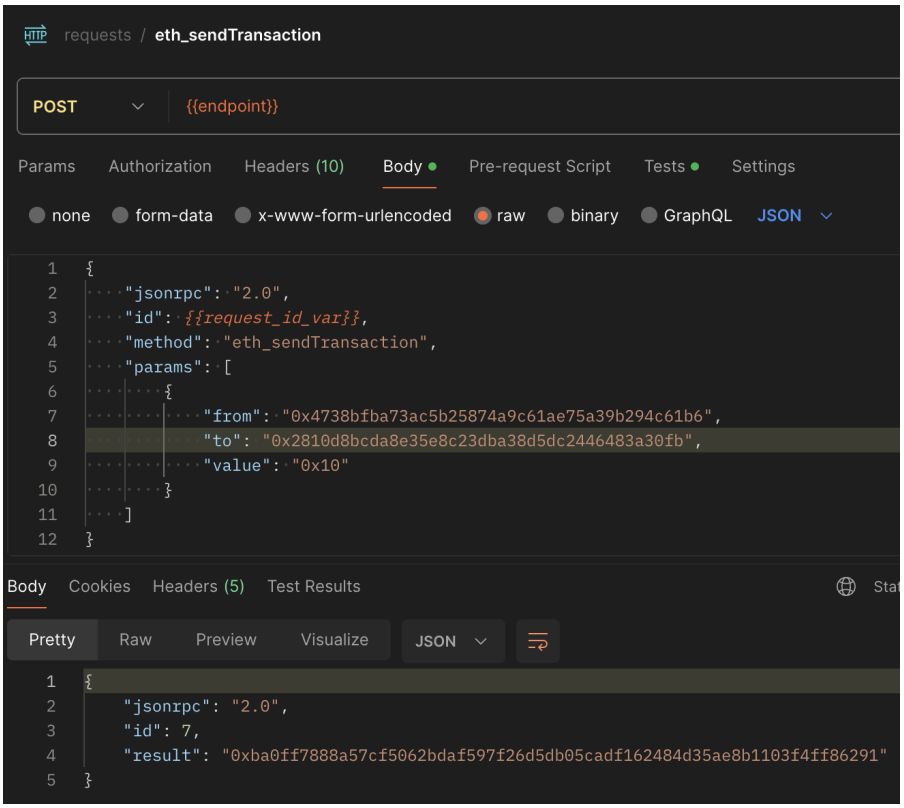
Створення нового аккаунту



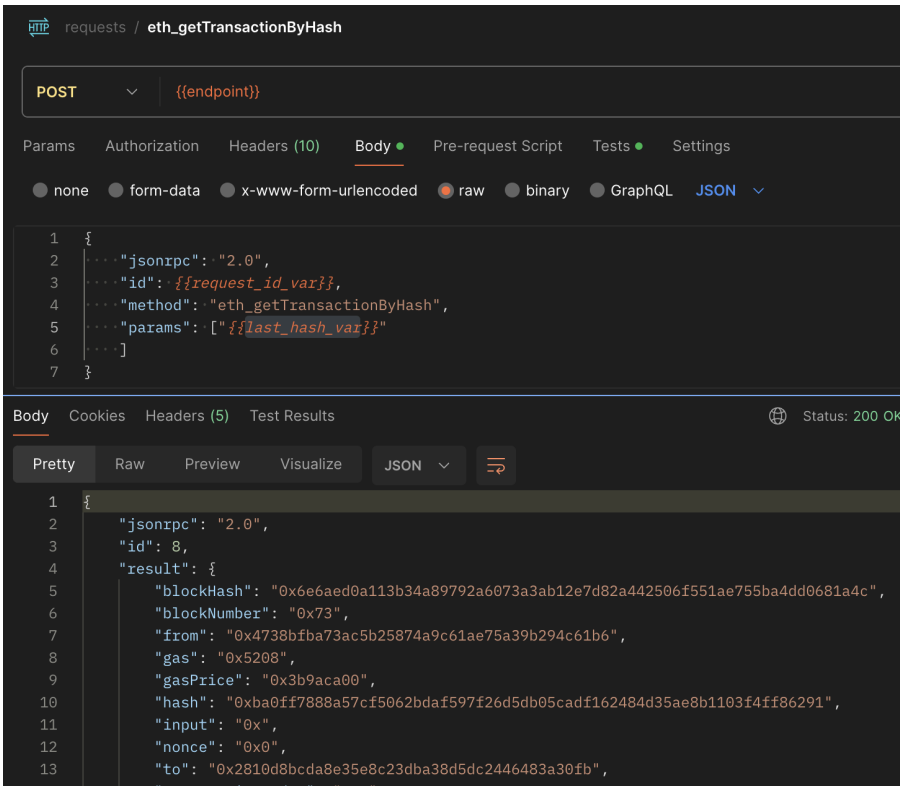
Розблокування аккаунту для переводу коштів



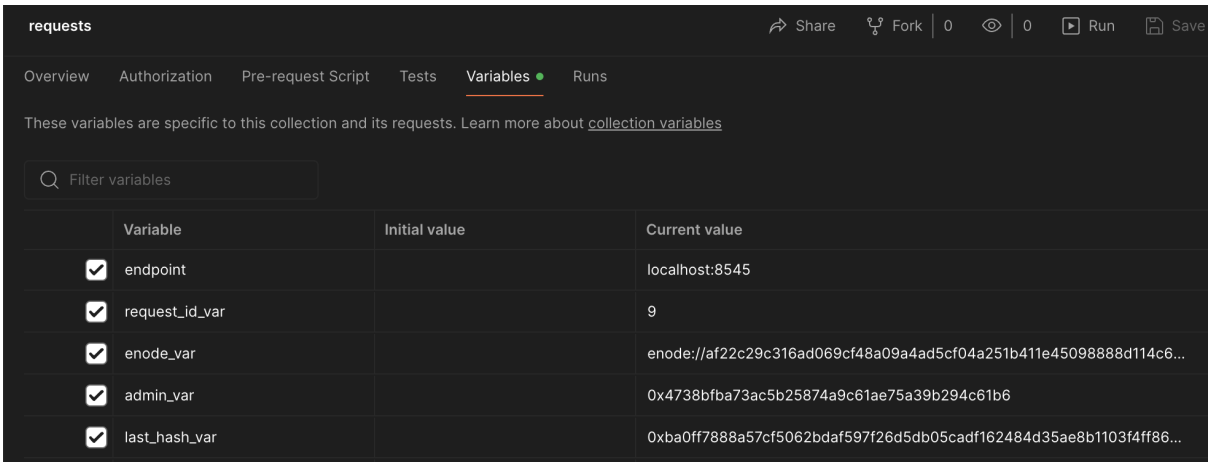
Створення транзакції



Перевірка транзакції за хешем



Кінцевий стан змінних



посилання

https://gist.github.com/0mkara/b953cc2585b18ee098cd
https://www.investopedia.com/news/what-byzantium-hard-fork-ethereum/
https://merehead.com/ua/blog/develop-private-ethereum-blockchain/
https://www.jsonrpc.org/specification
https://ethereum.org/en/developers/docs/apis/json-rpc/
https://www.researchgate.net/publication/340418164_Mathematics_and_Data_Structures_in_Blockchain_and_Ethereum
https://ethereum.org/en/whitepaper/