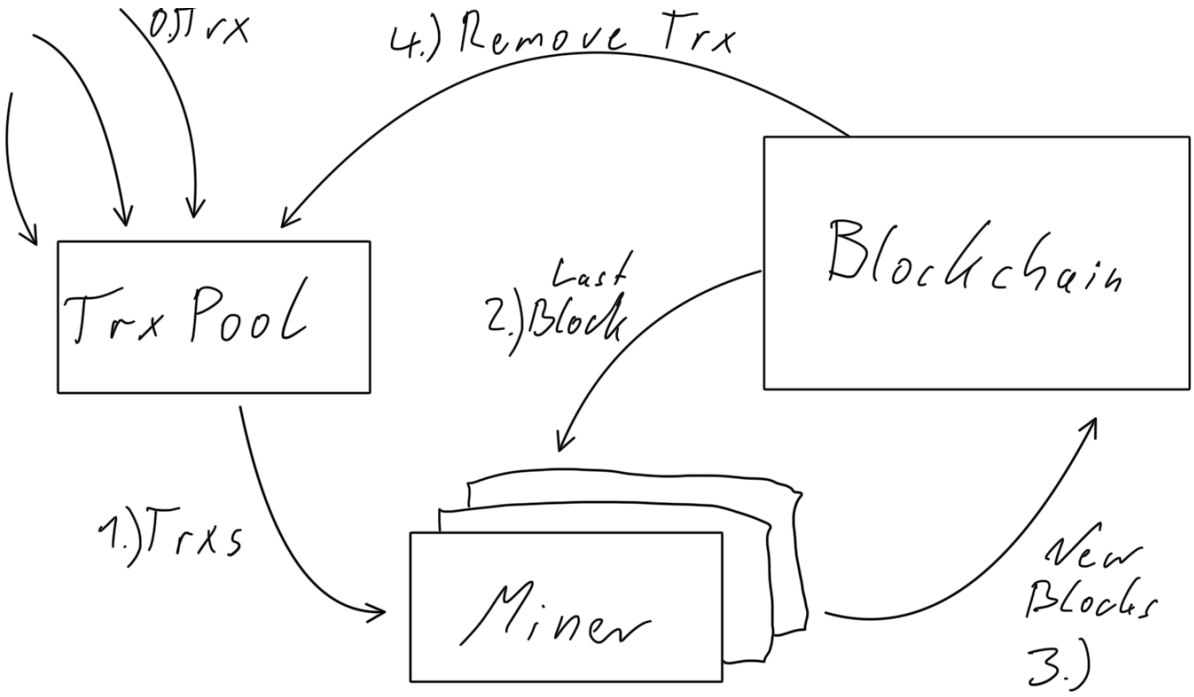
Blockchain Application

Tech Stack

- NodeJS, Express, Express-validator, nodemon
- REST
- Docker & Docker Compose
- Postman & Thunder Client

System Overview

This is a simple blockchain application that plays with the concepts of proof of work, transaction pools, and cryptographic signatures. On the infrastructure side, it uses docker containers and docker compose to scale certain components (mining).



Proof-of-Work Consensus

The system uses a proof-of-work consensus. This means that miners must perform a certain amount of (useless) work in order to commit a block to the blockchain. As long as the majority of miners abide by the system's rules, the system will remain in a valid state. If a malicious participant wants to manipulate the system, it is not enough to create many miners. He must provide them with more CPU resources than the sum of correct nodes.

Model

Transaction (Trx)

A transaction represents the record of a flow of money from one user to another. It therefore contains a creation timestamp, sender, receiver and an amount. It also contains a signature of the sender confirming that the transaction was created by him. A hash of the transaction is derived from these values, which can identify the transaction.

> ⚠️ **Transactions not validated:**
>
> The miners do not validate the transactions before mining a block as they would in a full blockchain system. Therefore, anyone can create transactions with any amount even if one does not have sufficient funds on the blockchain or use a signature that does not check out.

```
{
    "sender": "john",
    "receiver": "alice",
    "amount": 10,
    "signature": "asd23wea",
    "createdAt": 1646927117243,
    "hash": "73c9f1aedfd0bbbcf57410312b01e55da0a4d3f2d306f360ca12fa567de4143a"
}
```

Block

A block in the blockchain contains a timestamp of the block's creation time (start of mining), a set of transactions, the ID of the miner who mined the block. It also contains the hash value of the previous block and a nonce. A hash of the block is created from these values.

The process of mining is defined as the continuous effort of the miner to change the nonce so that the newly calculated hash of the block starts with a number of zeros (target). If the blockchain has a target of 6, it will only accept new blocks with a hash of 6 leading zeros (e.g., "000000b10640a0ae5c4...").

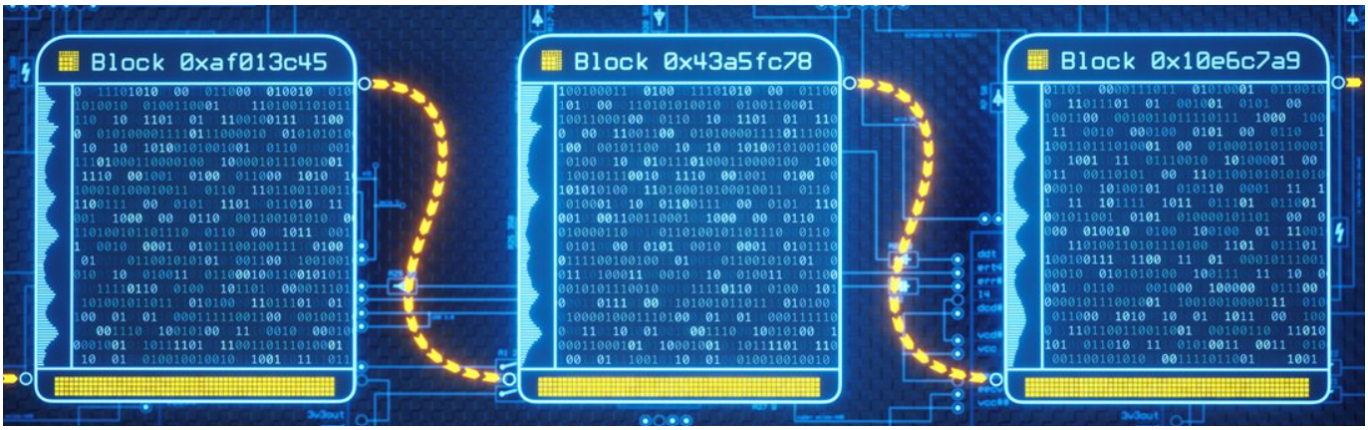
```
{
    "nonce": 128,
    "timestamp": 1646927117243,
    "trxs": [
      {
        "sender": "john",
        "receiver": "alice",
        "amount": 10,
        "signature": "asd23wea",
        "createdAt": 1646927117243,
        "hash": "73c9f1aedfd0bbbcf57410312b01e55da0a4d3f2d306f360ca12fa567de4143a"
      },
      // ...
    ],
```

```
    "prevHash": "000000de177b684a3ce6c202c5703fead8894fc35a8df43792b10640a0ae5c4",
    "minedBy": "987",
    "hash": "000000b10640a0ae5c46465ed4de177b684a3ce6c202c5703fead8894fc35a8"
}
```

The genesis block of the blockchain is a special block. It does not contain a "prevHash" value because it is the first block.

Immutability of the Blockchain

Since the hash of the previous block is part of a block, it is very difficult for an attacker to change anything in the blockchain. A change in a block (or in a trasanction within a block) would result in a changed hash. Moreover, the following block contains the hash of the changed block. Therefore, the hash of the following block must also be recalculated. This continues until the last commited block. Computing all these hashes requires a lot of resources. Meanwhile, legitimate miners mine based on the legitimate chain. Therefore, the attacker would need so many computational resources to repeat all the work done by the legitimate miners since the modified block and catch up. This may be possible if the block was added recently. However, the more blocks are appended after a block, the more difficult it becomes to change it.



Run Instructions

Production

Run

Run all modules. If it has not been run before, docker compose will build the services automatically. The number of miners can be arbitarily chosen.

```
docker-compose up --scale miner=3
```

Rebuild

If necessary to rebuild the services, use the following commands. Rebuild all services:

```
docker-compose build
```

Rebuild a single service:

```
docker-compose build [CONTAINER_NAME]
```

Development

To run the services without docker and with nodemon to enable hot-reloading and show changes immediately, run the services individually:

```
# Run blockchain
cd blockchain
npm install
npx nodemon

# Run trxpool
cd ../trxpool
npm install
npx nodemon

# Run miner
cd ../miner
npm install
npx nodemon
```

API

To interact and test the services, a REST-Client such as Postman or Thunder Client can be used. Configuration files are available in the directory .api_testing.

Blockchain

GET /target

Retrieve the target for the blockchain. The target is defined as the number of leading zeros of the hash of a new block in order to be accepted by the blockchain. This endpoint is queried by the miners so they can mine new blocks meeting the target.

```
{
    "target": 6
}
```

GET /chain/blocks

Retrieve all Blocks from the Blockchain

```
[
  {
    "nonce": 0,
    "timestamp": 1646927117243,
    "trxs": [
      {
        "sender": "john",
        "receiver": "alice",
        "amount": 10,
        "signature": "asd23wea",
        "createdAt": 1646927117243,
        "hash": "73c9f1aedfd0bbbcf57410312b01e55da0a4d3f2d306f360ca12fa567de4143a"
      },
      // ...
    ],
    "prevHash": "000000b10640a0ae5c46465ed4de177b684a3ce6c202c5703fead8894fc35a8",
    "minedBy": "876",
    "hash": "000000de177b684a3ce6c202c5703fead8894fc35a8df43792b10640a0ae5c4"
  },
  // ...
]
```

GET /chain/blocks/:index

Retrieve nth block from the blockchain.


```
{
    "nonce": 0,
    "timestamp": 1646927117243,
    "trxs": [
      {
        "sender": "john",
        "receiver": "alice",
        "amount": 10,
        "signature": "asd23wea",
        "createdAt": 1646927117243,
        "hash": "73c9f1aedfd0bbbcf57410312b01e55da0a4d3f2d306f360ca12fa567de4143a"
      },
      // ...
    ],
    "prevHash": "000000de177b684a3ce6c202c5703fead8894fc35a8df43792b10640a0ae5c4",
    "minedBy": "876",
    "hash": "000000b10640a0ae5c46465ed4de177b684a3ce6c202c5703fead8894fc35a8"
}
```


GET /chain/blocks/latest

Retrieve the last (most recently added) block from the Blockchain


```
{
    "nonce": 0,
    "timestamp": 1646927117243,
    "trxs": [
      {
        "sender": "john",
        "receiver": "alice",
        "amount": 10,
        "signature": "asd23wea",
        "createdAt": 1646927117243,
        "hash": "73c9f1aedfd0bbbcf57410312b01e55da0a4d3f2d306f360ca12fa567de4143a"
      },
      // ...
    ],
    "prevHash": "000000de177b684a3ce6c202c5703fead8894fc35a8df43792b10640a0ae5c4",
    "minedBy": "876",
    "hash": "000000b10640a0ae5c46465ed4de177b684a3ce6c202c5703fead8894fc35a8"
}
```

POST /chain/blocks

This endpoint is used by the Miners to send a new block to the blockchain. The blockchain service than validates that all the hashes are correctly calculated and that the proposed block refers to the latest block of the chain. If this is not the case, the block is rejected.

```
{
  "nonce": 78434,
  "timestamp": 1646918047703,
  "minedBy": "786",
  "trxs": [{
      "createdAt": 1646916914999,
      "sender": "Bernd",
      "receiver": "Sofie",
      "amount": "32",
      "signature": "asdwe",
      "hash": "d92a8f18f87c81cfd38f3008a9da3b934243ebe1759021a217541d7954d11b10"
    },
    // ...
  ],
  "prevHash": "a3513e1110acb9111579a8f9a49e3d6f5629d5248d314ba3a7f5643740137ab6",
  "hash": "00000e4b2469446305586c043663aaed50d1c7b826b2a44358264278471c5a7c"
}
```

TransactionPool

The transaction pool service receives the transactions from the users of the system. Without the transaction pool the users would need to supply their transactions to all the miners individually. Before the beginning of the mining, the miner queries the transactionpool for all open transactions.

POST /trxpool

Add a new transaction to the pool. The timestamp and hash a created by the pool. If a transaction with the same hash already exists, an HTTP error is returned.

```
{
    "sender": "Bernd",
    "receiver": "Sofie",
    "amount": 32,
    "signature": "sadjlsjdlakds"
}
```

The endpoint returns the created transaction (including the timestamp and the hash).

GET /trxpool

Retrieve an array of all transactions in the pool

```
[
  {
    "createdAt": 1646930329740,
    "sender": "Bernd",
    "receiver": "Sofie",
    "amount": 32,
    "signature": "",
    "hash": "04ec8834e59f2be059982b65a8d5b3e5c770878db98153e193cf07e8e6945aff"
  },
  {
    "createdAt": 1646930330817,
    "sender": "Bernd",
    "receiver": "Sofie",
    "amount": 32,
    "signature": "",
    "hash": "60f528f7fb6ac5bb1e75267325c9b05e804ef1755e6ed6fdd3f538bbd929b96c"
  },
  // ...
]
```

DELETE /trxpool/:hash

Delete the transaction with the hash. Fails if no transaction with the hash is in the pool. If deleted successfully, the endpoint returns the transaction.