



CMSE353 Security of Software Systems
Term Project Report

Team members

Seyit Ahmet Inci – 19331143

Sinem Imge Turgut - 19001316

Emre Çakır – 20001797

Jamshid Artykov – 19700076

Group N0: 02

Term Project Topic: Blockchain

Semester Term: 2021-2022 Fall Term

Contents

Problem Definition	3
Teamwork Distribution and Meetings	3
Meetings	3
Word break-down	4
Algorithms used	4
Database Structure	5
Implementation Tools	5
Program function and class descriptions	6
blockchain/block.js	6
blockchain/index.js	6
app/p2p-server.js	7
app/index.js	7
app/miner.js	7
wallet/index.js	7
wallet/transaction.js	8
wallet/transactionPool.js	9
chain-util.js	9
config.js	9
User Guide	10
Conducted Tests	15
References	18

Problem Definition

The blockchain is a distributed and decentralized ledger that stores data such as a transaction, and that is publicly shared across all the nodes of its network. The blockchain consists of a collection of blocks and each block acts as a storage unit. Blockchain technology can be used in cryptocurrency, healthcare, supply chain and many more business types.

Teamwork Distribution and Meetings

Meetings

Date	Meeting No	Participants	What was discussed
18.12.2021	1	Emre, Seyit, Sinem, Jamshid	Finding related open-source code implementation or online course for implementing your own blockchain.
25.12.2021	2	Emre, Seyit, Sinem, Jamshid	Work break-down and distribution of tasks.
02.01.2022	3	Emre, Seyit, Sinem, Jamshid	Finalization of backend application.
08.01.2022	4	Emre, Seyit, Sinem, Jamshid	Testing, documentation, and finalization of work.

Word break-down

	Sinem	Emre	Seyit	Jamshid
Blocks				
Blockchain				
API and Network				
Proof of Work				
Wallet and Transactions				
Transaction Pool				
Mine Transactions				

Algorithms used

SHA-256 Hashing alorithm: SHA-256 is a cryptographic hash function. Several cryptocurrencies use this algorithm for verifying transaction and calculating proof of work. The algorithm is included inside of java built-in security library.

JSON.stringify: Algorithm is used to convert data to string before sending data in to web server.

Secp256k1: Secp256k1 is the name of the elliptic curve used by Bitcoin to implement its public key cryptography.

Database Structure

The blockchain data structure is a back-linked list of blocks of data such as transactions, which is ordered. Each block is identifiable by a hash, generated using the cryptographic hash algorithm on the header of the block. Each block has timestamp, data, hash of the current block and references to a previous block, also known as the parent block, in the “previous block hash” field, in the block header.



Implementation Tools

The application uses JavaScript as a backend language for our application. The code is written with Visual Studio Code Editor. To be able to run the code user must install latest version of node.js and Postman to interact with an application. We implemented the system with unit test cases using test environment ‘jest’.

Program function and class descriptions

blockchain/block.js

class Block: Block class with constructor. Each block has timestamp, previous hash, hash, data, nonce and difficulty.

toString(): Returns formatted version of the block.

genesis(): First block in blockchain is called genesis block. Used to initialize blockchain.

mineBlock(lastblock, data): Generating a block is equated to an act of mining since it takes computational power to “mine” the block. The function takes lastblock data to calculate its last hash.

hash(timestamp, lastHash, data, nonce, difficulty): Uses SHA256 algorithm from chain-util.js file to hash the given data and returns string version of it with the help of toString() algorithm.

blockHash(block): Hashes the given block inside of function.

adjustDif(lastblock,currentTime): The function is used to adjust difficulty of hashing. The higher difficulty is the longer it takes to mine new block. Difficulty is adjusted based on the how fast it takes to mine new block. If it's fast, increase difficulty and if it's slow decrease difficulty.

blockchain/index.js

class Blockchain: Blockchain class with constructor. The chain starts with genesis block.

addBlock(data): Add block to the blockchain once it has been mined.

isValidChain(chain): Ensures that the chain is not corrupt once there are multiple contributors to the chain. First it checks if the blockchain starts with the genesis block. Then it checks if the hashes are properly generated.

replaceChain(newChain): Replaces the current chain if another contributor submits a valid chain. Only replaces chains that are longer than current chain

app/p2p-server.js

In this file we implemented Websocket module. This will allow us to create real-time connections between multiple users of the blockchain. The same class that creates the original websocket server will be used to connect to existing servers using **connectToPeers()** functions. **messageHandler(socket)** is used to send messages between sockets. To synchronize the chain accross all users **replaceChain** function will be used.

app/index.js

By using “express” module we can create express application with a lot of functionality. With this application we can post and get data from our code using Postman. Every time when blockchain application starts, this file creates new instances of Blockchain, Wallet, TransactionPool, Miner and P2Pserver.

app/miner.js

class Miner: Miners are user types that have ability to confirm transactions from transaction pool and add them to the blockchain. So each user can become miner.

mine(): Check the validity of transaction using **validTransaction** function, if the transactions are valid add the transactions to the blockchain. Once added, synchronize the chain accross users. Clear transaction pool and broadcast it the all the users so same transaction is not mined twice. Also, reward for the miner is included here.

wallet/index.js

class wallet: Wallets contain balance, public key and private key to sign the transaction.

toString(): Formatting and printing Wallet details.

sign(datahash): Takes any data in its hash form, and returns a signature based on the keyPair.

createTransaction(recipient,amount,blockchain,transactionPool): Method is used to create transactions. Method checks if the amount exceed balance and also it will check if the transaction exists by the sender in the transactionPool. If it exists updates existing transaction.

blockchainWallet(): Creates system wallet in order to approve mining reward transactions.

calculateBalance(blockchain): Calculates balance by iterating over every existing transactions from begining.

wallet/transaction.js

class Transaction: Transactions represent exchanges in cryptocurrency. Each transaction has an input field which provides information about the sender of the transaction, output fields which detail how much currency the sender is giving to other wallets and a unique id.

newTransaction(senderWallet, recipient,amount): Creates new transaction. Also checks if the user is trying to send more money than they have.

signTransaction(transaction,senderWallet): Signs the transaction with timestamp, amount, address and with signature of the sender.

verifyTransaction(transaction): Transactions generate signatures based upon their outputs and their private keys. This functions provides a way to verify authenticity of those signature using verifySignature function.

update(senderWallet,recipient,amount): This fuction handles the addition of new output objects to an existing transaction.

rewardTransaction(mineWallet,blockchainWallet): Sets miners to recieve rewards for mining. The reward transactions are given and signed by system wallet.

wallet/transactionPool.js

class transactionPool: A transaction pool will collect all transactions submitted by individuals in the cryptocurrency network. Then, miners will do the work of taking transactions from the pool and including them in the blockchain. This class represent transactions in an array.

exisitingTransaction(address): Checks if the address has an existing transaction in the pool.

validTransactions(): This function checks validity of transaction. Transactions must pass 2 conditions. First, its total output amount matches the original balance specified in the input amount. Second, we'll also verify the signature of every transaction to make sure that the data has not been corrupted after it was sent by the original sender.

clear(): Clears the transaction pool once mined by the miner.

chain-util.js

class chainUtil: This class contains methods for assisting application with different functionalities.

genKeyPair(): Generates public and private key pairs using secp256k1 algorithm.

id(): Generates unique id using uuidV1 algorithm.

hash(data): hashes given data using SHA256 algorithm

verifySignature(publicKey,signature,datahash): Verifies signature based on public key, signature and datahash.

config.js

This file contains hardcoded data that is used accross application. Values can be changed to experiment with application.

User Guide

Prerequisites

Node.js: <https://nodejs.org/en/download/>

Postman: <https://www.postman.com/downloads/>

Git-bash: <https://git-scm.com/downloads>

How to run

1. Extract Blockchain353 folder into workspace
2. Open Git Bash and type: `cd "{path to blockchain353 folder}"` with double quotes on

For instance it should like this:




```
MINGW64:/c/Users/DC/Desktop/Blockchain353

DC@LAPTOP-HT9NTNSJ MINGW64 ~
$ cd "C:\Users\DC\Desktop\Blockchain353"

DC@LAPTOP-HT9NTNSJ MINGW64 ~/Desktop/Blockchain353 (master)
$ |
```

3. Type: **npm start** to start application. The Git Bash should run while using the app!



```
DC@LAPTOP-HT9NTNSJ MINGW64 ~/Desktop/Blockchain353 (master)
$ npm start

> bc-learn@1.0.0 start
> node ./app

Listening for peer-peer 5001
App Listening to 3001
```

4. Open new Git Bash in a new window. Type:

HTTP_PORT=3002 P2P_PORT=5002 PEER=ws://localhost:5001 npm start

```

DC@LAPTOP-HT9NTNSJ MINGW64 ~/Desktop/Blockchain353 (master)
$ HTTP_PORT=3002 P2P_PORT=5002 PEER=ws://localhost:5001 npm start

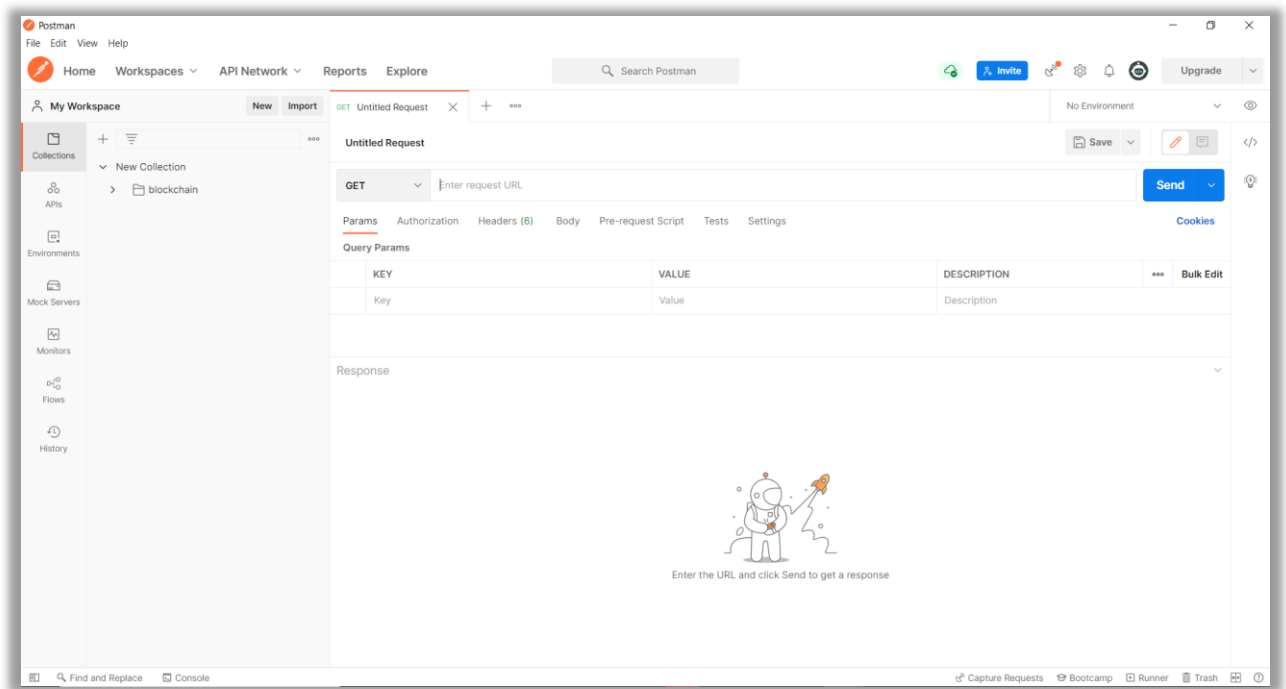
> bc-learn@1.0.0 start
> node ./app

Listening for peer-peer 5002
App Listening to 3002

```

This will add another node to our application. Now we have 2 nodes in different ports. To add more users, open new Git Bash window and change values in HTTP_PORT and P2P_PORT. Don't forget to change directories with every new Git Bash window.

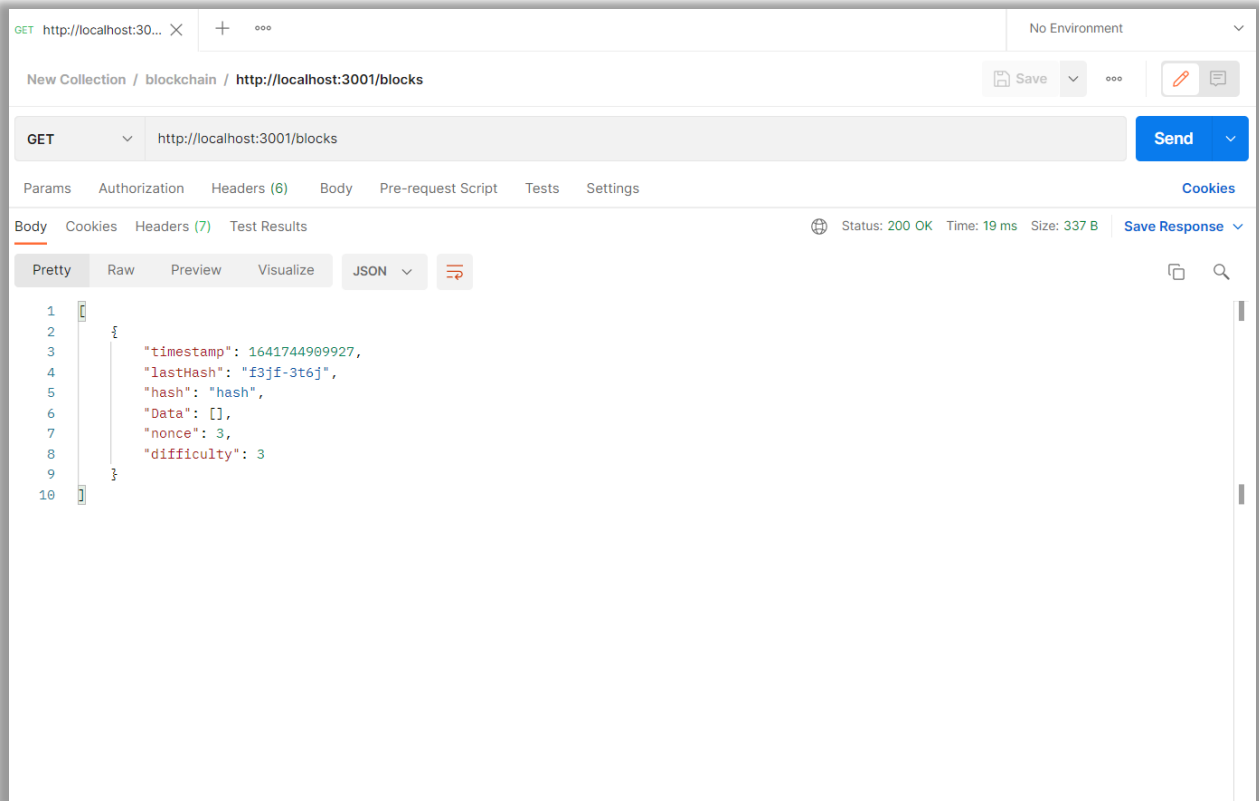
5. Open postman. Login or register with new account. Once logged in head to Workspaces



6. Here is the list of actions. Use connected port numbers instead of {HTTP_PORT} to switch between users. Click "Send" to perform action.

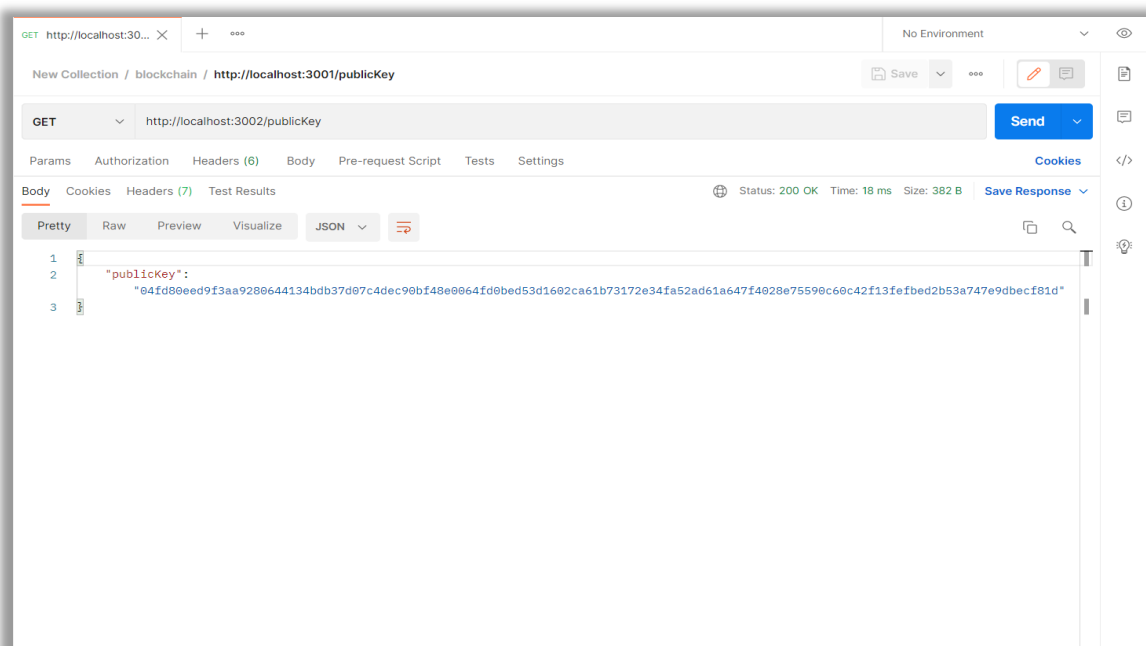
- **GET method:** http://localhost:{HTTP_PORT}/blocks

Prints out blockchain. The first block contains genesis block information.



- **GET method:** `http://localhost:{HTTP_PORT}/publicKey`

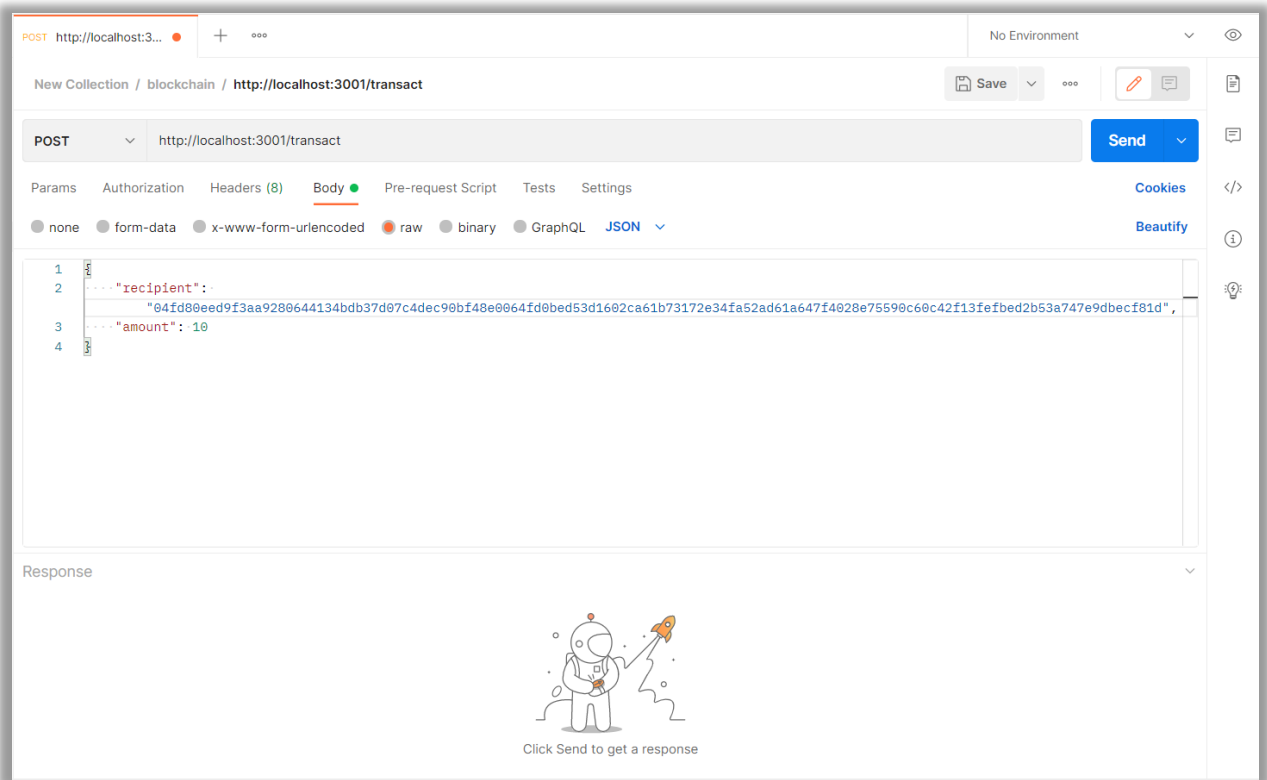
Prints public key or address of the connected node.



- **POST method:** `http://localhost:{HTTP_PORT}/transact`

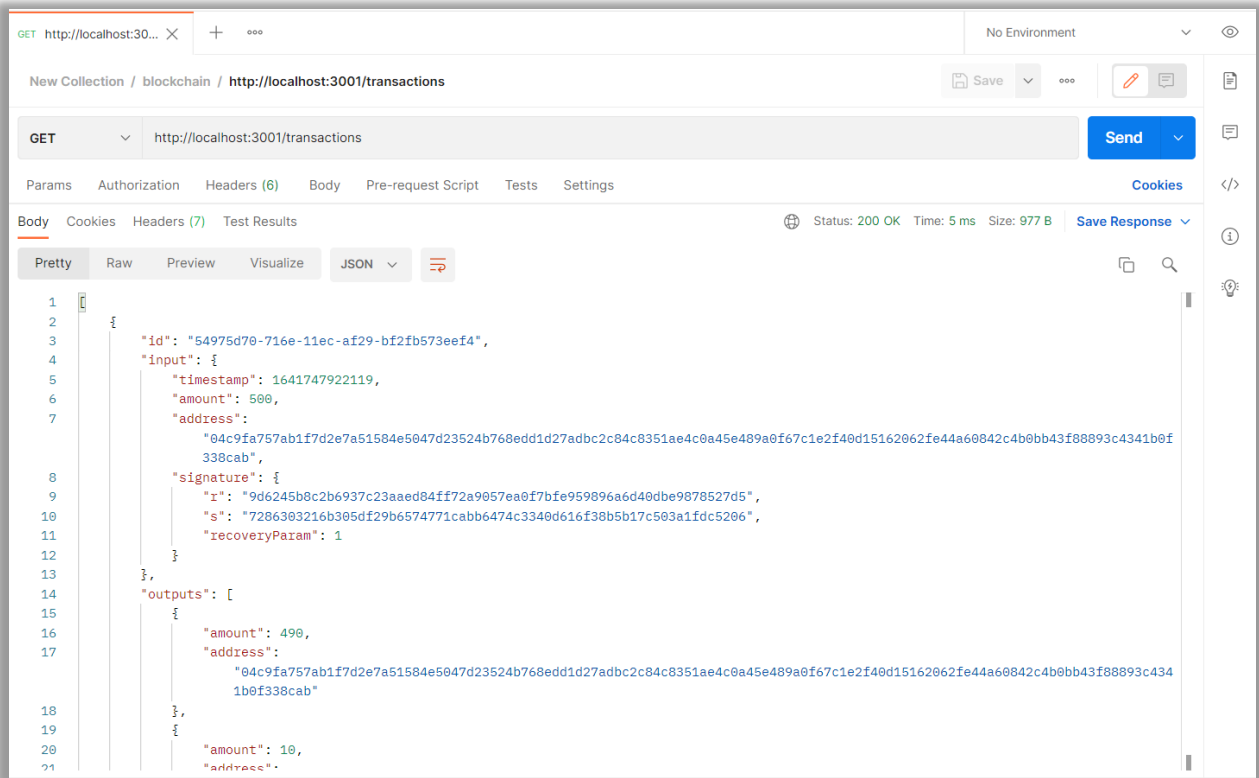
Initialize transaction with the recipient address and amount to be sent. Change GET to POST. Click on “Body”, and select “raw”, then change TXT to JSON. Once everything is set up type following command as shown in the picture:

```
{ "recipient": "[publicKey]", "amount": [amount] }
```



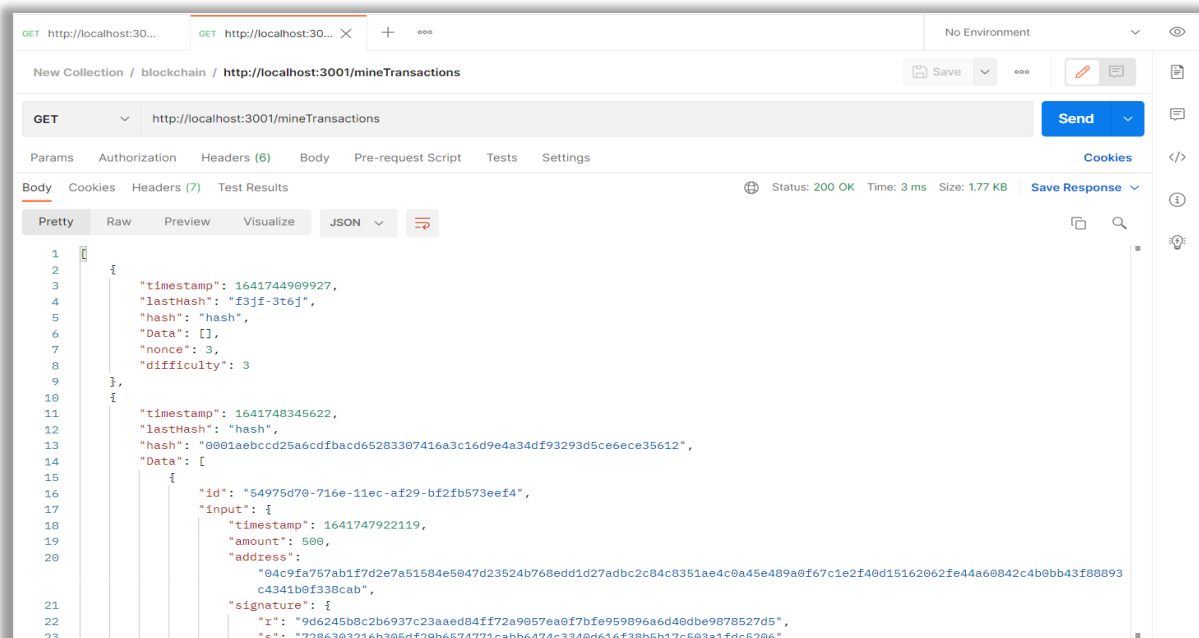
- **GET method:** `http://localhost:{HTTP_PORT}/transactions`

Prints transactions waiting in transaction pool

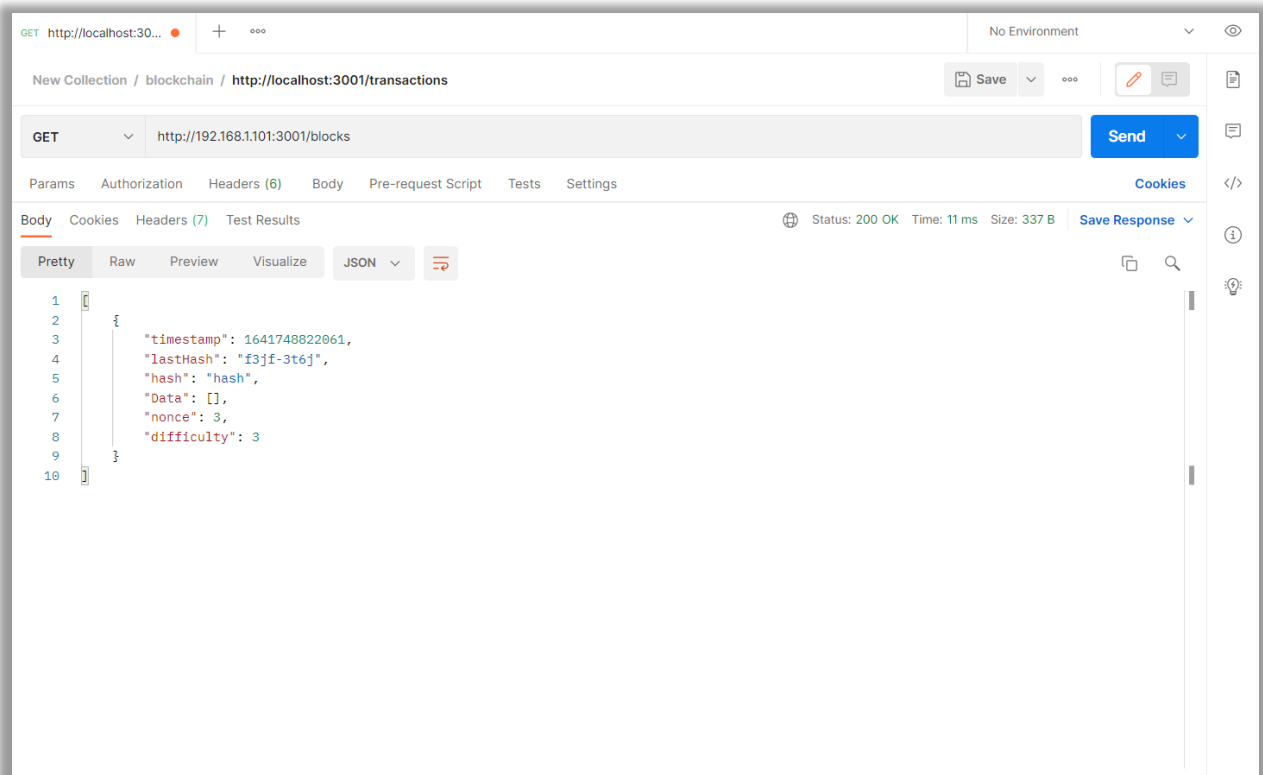


- **GET method:** `http://localhost:{HTTP_PORT}/mineTransaction`

Mines available transactions in transaction pool, adds transactions as a block and redirects to /blocks which prints out blockchain



7. Once finished, terminate ports with CTRL+C command in Git Bash.
8. The application only works on the same network. If you want to test application from different computer, you must have Postman on that computer. The user from different computer must enter IPv4 address instead of localhost. For example it should look like this `http://192.168.1.101:3002`



Conducted Tests

Test Case 1

Test Case: Creating node on default node and connecting another node to peer-2-peer port 5001.

Result: Successfully connected.

```

DC@LAPTOP-HT9NTNSJ MINGW64 ~/Desktop/Blockchain353 (master)
$ npm start

> bc-learn@1.0.0 start
> node ./app

Listening for peer-peer 5001
App Listening to 3001

```

```

DC@LAPTOP-HT9NTNSJ MINGW64 ~/Desktop/Blockchain353 (master)
$ HTTP_PORT=3002 P2P_PORT=5002 PEER=ws://localhost:5001 npm start

> bc-learn@1.0.0 start
> node ./app

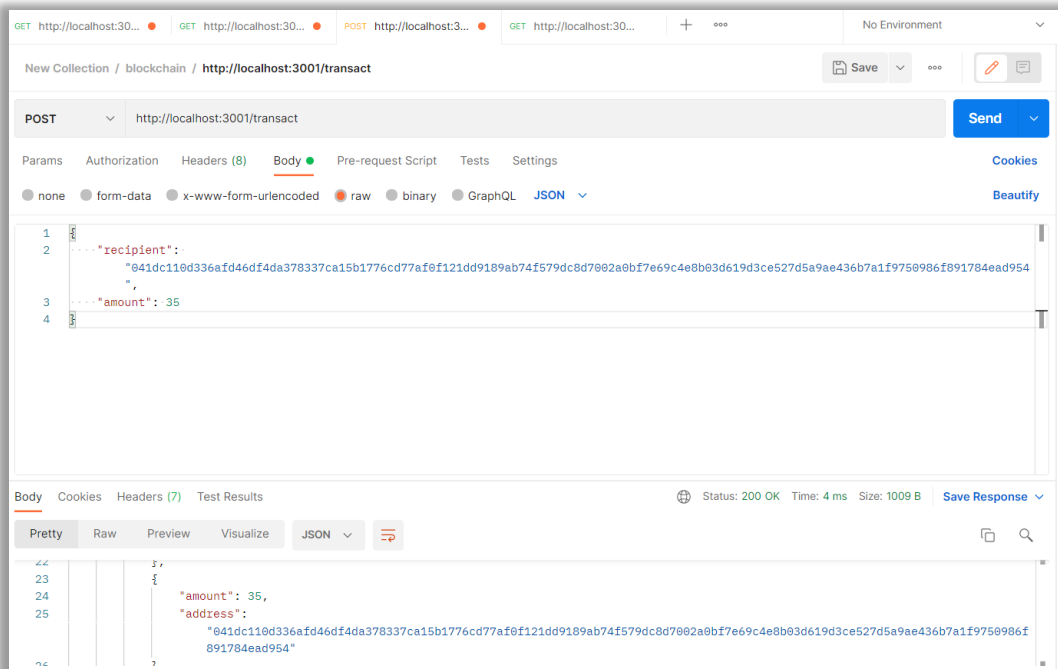
Listening for peer-peer 5002
App Listening to 3002

```

Test Case 2

Test Case: Node on port 3001 can create transaction and send 35 coins using publicKey of 3002

Result: Successfully sent.



Test Case 3

Test Case: Node on port 3002 can mine pending transactions from transactions pool.

Result: Successful.

GET

http://localhost:3002/mineTransactions

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 3 ms

Size: 1.8 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "timestamp": 1641760916853,
3    "lastHash": "f3jf-3t6j",
4    "hash": "hash",
5    "Data": [],
6    "nonce": 3,
7    "difficulty": 3
8  },
9  {
10   "timestamp": 1641762242561,
11   "lastHash": "hash",
12   "hash": "00cc249dd977e7bfd40d91bb896ca5d316ae15cffa8456fe00cfae6dbd6bd7ba",
13   "Data": [
14     {
15       "id": "bdd9f220-718c-11ec-b9ad-3f79572401e7",
16       "input": {
17         "timestamp": 1641762021103,
18         "amount": 500,
19         "address":
20           "047b555fa43a0a506ea76df0ebb6f716131b7c9cf0827664c043488590817e19981640ddb96f87fdd9ebad25c037981211be8f7ed8105bbbf6181c567728968291",
21       "signature": {
22         "r": "aac8c7396f22039a47face44c3d7caf5b51ba570d825ba55020eb1e8cf6227f1",
23         "s": "f1bb96c6af4524a8218c757c2e4b1924e4219e67527ea05888556b5656731ce4",
24         "recoveryParam": 0
25       }
26     },
27     "outputs": [
28       {
29         "amount": 455,
30         "address":
31           "047b555fa43a0a506ea76df0ebb6f716131b7c9cf0827664c043488590817e19981640ddb96f87fdd9ebad25c037981211be8f7ed8105bbbf6181c567728968291"
32       },
33       {
34         "amount": 10,
35         "address": "test3"
36       },
37       {
38         "amount": 35,
39         "address":
40           "041dc110d336afd46df4da378337ca15b1776cd77af0f121dd9189ab74f579dc8d7002a0bf7e69c4e8b03d619d3ce527d5a9ae436b7a1f9750986f891784ead954"
41       }
42     ]
43   },
44   "nonce": 327,
45   "difficulty": 2
46 }
}
```

Test Case 4

Test Case: Node on port 3001 can view updated blockchain

Result: Successful.

GET http://localhost:3002/blocks

Status: 200 OK Time: 4 ms Size: 1.8 KB

```
1 {
2   {
3     "timestamp": 1641760916853,
4     "lastHash": "f3jfi-3t6j",
5     "hash": "hash",
6     "Data": [],
7     "nonce": 3,
8     "difficulty": 3
9   },
10  {
11    "timestamp": 1641762242561,
12    "lastHash": "hash",
13    "hash": "00cc249dd977e7bfd40d91bb896ca5d316ae15cff8456fe00cfae6dbd6bd7ba",
14    "Data": [
15      {
16        "id": "bdd9f220-718c-11ec-b9ad-3f79572401e7",
17        "input": {
18          "timestamp": 1641762021103,
19          "amount": 500,
20          "address":
21            "047b555fa43a0a506ea76df0ebb6f716131b7c9cf0827664c043488590817e19981640ddb96f87fdd9ebad25c037981211be8f7ed8105bbb6181c567728968291",
22          "signature": {
23            "x": "aac8c7396f22039a47face44c3d7caf5b51ba570d825ba55020eb1e8cf6227f1"
24          }
25        }
26      }
27    ]
28  }
29 }
```

References

<https://github.com/15Dkatz/sf-chain-guides>

<https://www.udemy.com/course/build-blockchain/>