CRYPTOGRAPHY Project

Project Description:

We have implemented a Decentralized Stock Trade Management System in a web-based blockchain application that is designed to facilitate secure and transparent capture of stock transactions without the need for intermediaries. Using SHA-256 for proof of work, this system ensures the integrity and immutability of trade records while providing users with a decentralized platform for managing their accounts. We generate a RSA key pair(private and public keys) for every user and when a user is added to the network, its url and public key is shared with every user already registered in the network. Challenge Response Authentication is done using the SHA-256 algorithm of the HMAC library.
All the functionalities of this system are executed using API endpoints. We have used NodeJS for the base of the blockchain and ExpressJS along with POSTMAN for all the features and decentralized actions in the system.
According to the stage3 evaluation criteria for this project, we have implemented the 4 required functionalities as follows:

- createNewBlock()
- verifyTransaction()
- ‘/mine’ endpoint (for proof of work)
- ‘/address/:address’ endpoint (to view user), where :address is the address(url) of the user that we want to view. Here you can also view the Miner and how much reward(in BTC) it has earned till now.

We have implemented this project following some of the features of the Bitcoin crypto currency, like the transaction mining reward for a miner(also a transaction itself) is added to the next block, and the mining reward is 6.25 BTC as of 2024(following the “bitcoin halving” principle).

How to Run:

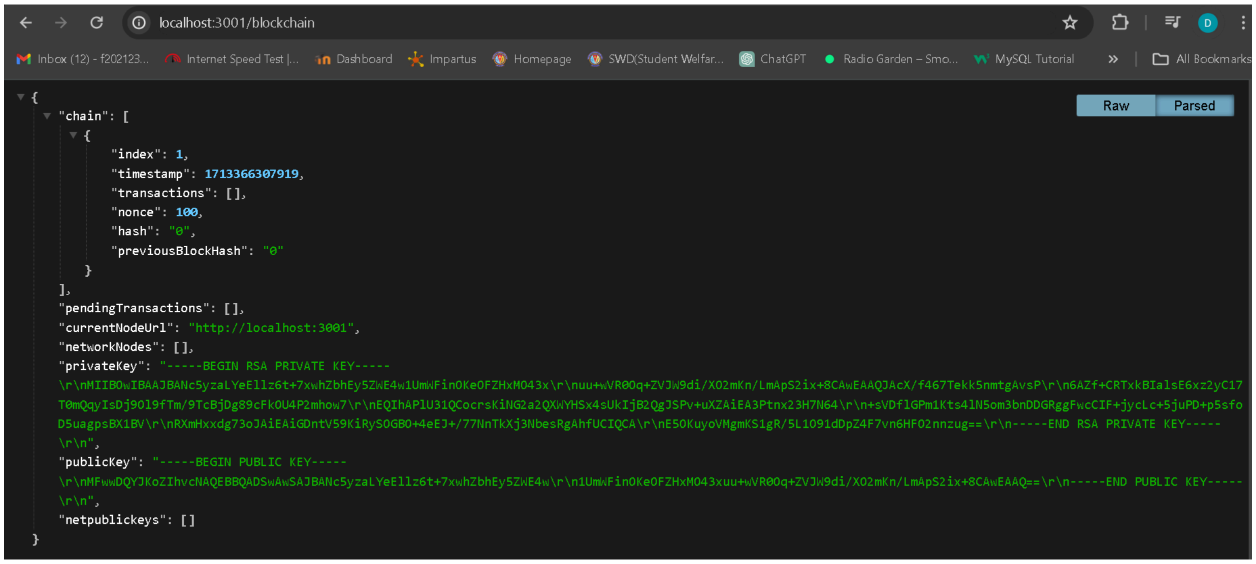
You need to download NodeJS, ExpressJS, and POSTMAN(for POST API).
Next you need to install a few packages, so open the terminal in the directory where you have stored this project and run these commands:

```
npm init
npm install sha256
npm install express
npm install nodemon
npm install body-parser
npm install uuid
npm install request-promise
npm install node-forge
```

In our project, we have defined users as nodes in the network. You can view which all nodes can be used in the package.json file and you can add more if you want by following the same syntax.

To run the code, type `npm run node_1` in the terminal to activate the user1 instance. Similarly you can run other users by using the same command for their nodes.

If you head over to http://localhost:3001/blockchain you can see the user as:

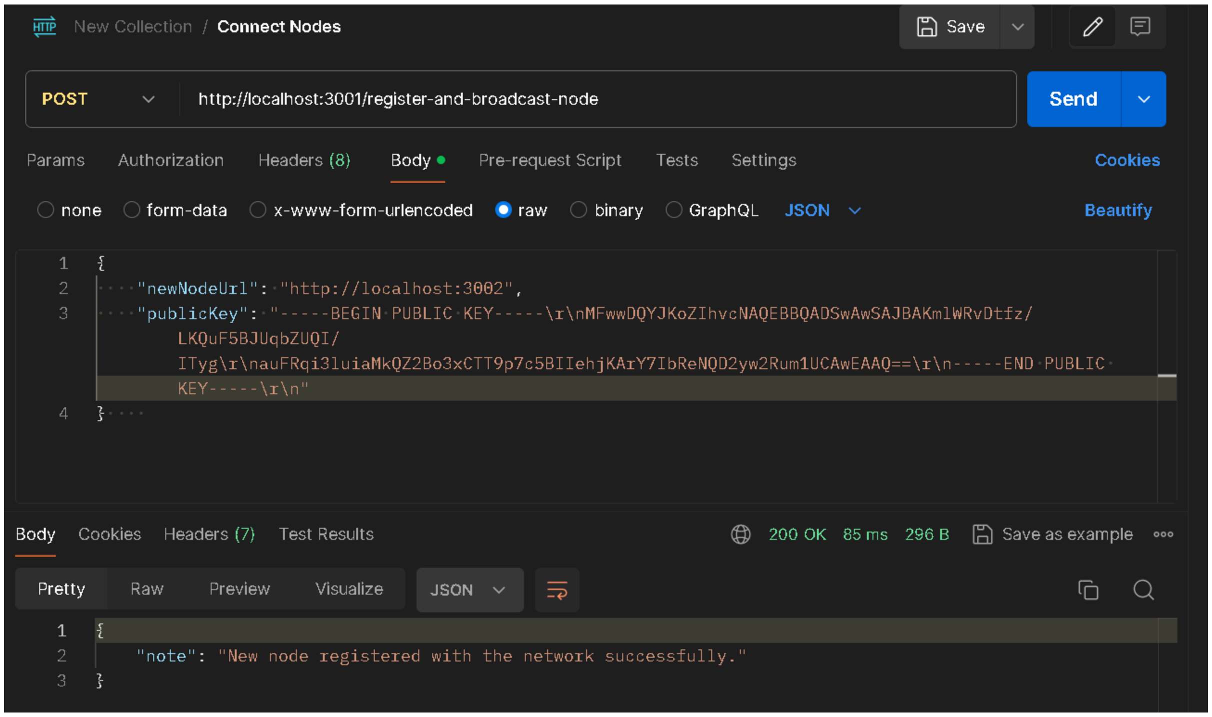


```
{
  "chain": [
    {
      "index": 1,
      "timestamp": 1713366307919,
      "transactions": [],
      "nonce": 100,
      "hash": "0",
      "previousBlockHash": "0"
    }
  ],
  "pendingTransactions": [],
  "currentNodeUrl": "http://localhost:3001",
  "networkNodes": [],
  "privateKey": "-----BEGIN RSA PRIVATE KEY-----
\r\nMIIBOwIBAAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w1UmWFinOKeOFZHxMO43x\r\nuu+wVR0Oq+ZVJW9di/XO2mKn/LmApS2ix+8CAwEAAQJAcX/f467Tekk5nmtgAvsP\r\n6AZf+CRTxkBIalsE6xz2yC17
T0mQqyIsDj9Ol9fTm/9TcBjDg89cFkOU4P2mhow7\r\nEQIhAPlU31QCocrsKiNG2a2QXWYHSx4sUkIjB2QgJSPv+uXZAiEA3Ptnx23H7N64\r\n+sVDflGPm1Kts4lN5om3bnDDGRggFwcCIF+jycLc+5juPD+p5sfo
D5uagpsBX1BV\r\nRXmHxxdg73oJAiEAiGDntV59KiRySOGBO+4eEJ+/77NnTkXj3NbesRgAhfUCIQCA\r\nE5OKuyoVMgmKS1gR/5L1O91dDpZ4F7vn6HFO2nnzug==\r\n-----END RSA PRIVATE KEY-----
\r\n",
  "publicKey": "-----BEGIN PUBLIC KEY-----
\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w\r\n1UmWFinOKeOFZHxMO43xuu+wVR0Oq+ZVJW9di/XO2mKn/LmApS2ix+8CAwEAAQ==\r\n-----END PUBLIC KEY-----
\r\n",
  "netpublickeys": []
}
```

The user has a genesis block created and it will add all other blocks to this.

Similarly you can do for node_2(port 3002).

Now to connect node_1 and node_2, you make a POST request on POSTMAN as follows:



POST http://localhost:3001/register-and-broadcast-node

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

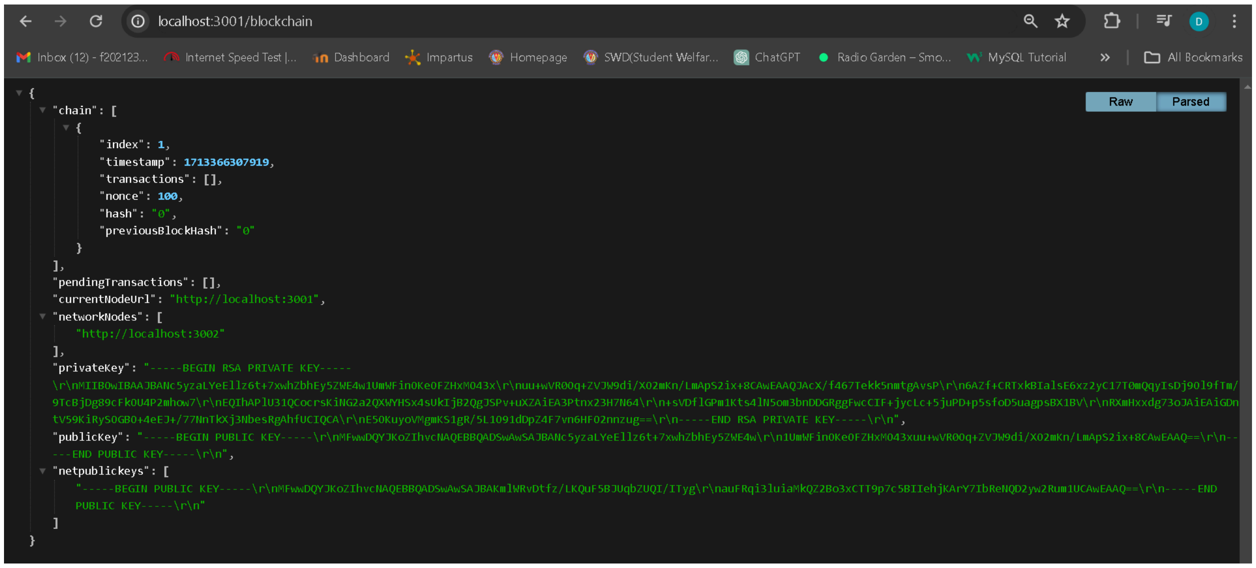
```
{
    "newNodeUrl": "http://localhost:3002",
    "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKmlWRvDtfz/
        LKQuF5BJUqbZUQI/
        ITyg\r\nauFRqi3luiaMkQZ2Bo3xCTT9p7c5BIIehjKArY7IbReNQD2yw2Rum1UCAwEAAQ==\r\n-----END PUBLIC
        KEY-----\r\n"
}
```

Body 200 OK 85 ms 296 B JSON

```
{
    "note": "New node registered with the network successfully."
}
```

On successful registration of new node(user), you will get the Response as mentioned above in the pic.

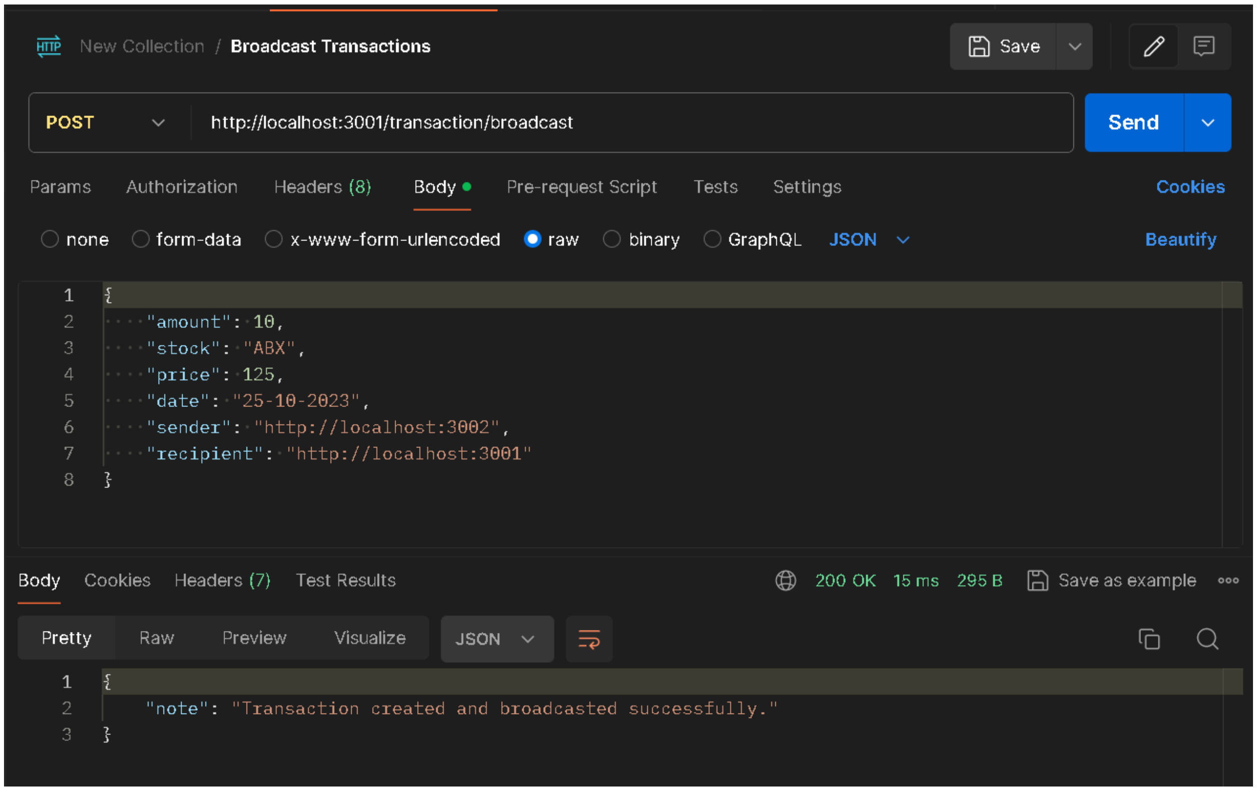
Now the user1 will look like:



```
{
  "chain": [
    {
      "index": 1,
      "timestamp": 1713366307919,
      "transactions": [],
      "nonce": 100,
      "hash": "0",
      "previousBlockHash": "0"
    }
  ],
  "pendingTransactions": [],
  "currentNodeUrl": "http://localhost:3001",
  "networkNodes": [
    "http://localhost:3002"
  ],
  "privateKey": "-----BEGIN RSA PRIVATE KEY-----
  \r\nMIIBOwIBAAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w1UmWFin0KeOFZHxMO43x\r\nuu+wVR0Oq+ZVJW9di/XO2mKn/LmApS2ix+8CAwEAAQJAcX/f467Tekk5nmtgAvsP\r\n6AZf+CRTxkBIalsE6xz2yC17T0mQqyIsDj9Ol9fTm/
  9TcBjDg89cFkOU4P2mhow7\r\nEQIhAPlU31QCocrsKiNG2a2QXWYHSx4sUkIjB2QgJSPv+uXZAiEA3Ptnx23H7N64\r\n+sVDflGPm1Kts4lN5om3bnDDGRggFwcCIF+jycLc+5juPD+p5sfoD5uagpsBX1BV\r\nRXmHxxdg73oJAiEAiGDn
  tV59KiRyS0GB0+4eEJ+/77NnTkXj3NbesRgAhfUCIQCA\r\nE5OKuyoVMgmKS1gR/5L1091dDpZ4F7vn6HFO2nnzug==\r\n-----END RSA PRIVATE KEY-----\r\n",
  "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w\r\n1UmWFin0KeOFZHxMO43xuu+wVR0Oq+ZVJW9di/XO2mKn/LmApS2ix+8CAwEAAQ==\r\n--
  ---END PUBLIC KEY-----\r\n",
  "netpublickeys": [
    "-----BEGIN PUBLIC KEY-----\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKmlWRvDtfz/LKQuF5BJUqbZUQI/ITyg\r\nauFRqi3luiaMkQZ2Bo3xCTT9p7c5BIIehjKArY7IbReNQD2yw2Rum1UCAwEAAQ==\r\n-----END
    PUBLIC KEY-----\r\n"
  ]
}
```

Here netpublickeys array contains the public keys of all other nodes connected to the its user, and the networkNodes array will contain the url of all the nodes currently connected in the network.

Now to add a transaction, we make a POST request in POSTMAN as follows:



POST http://localhost:3001/transaction/broadcast

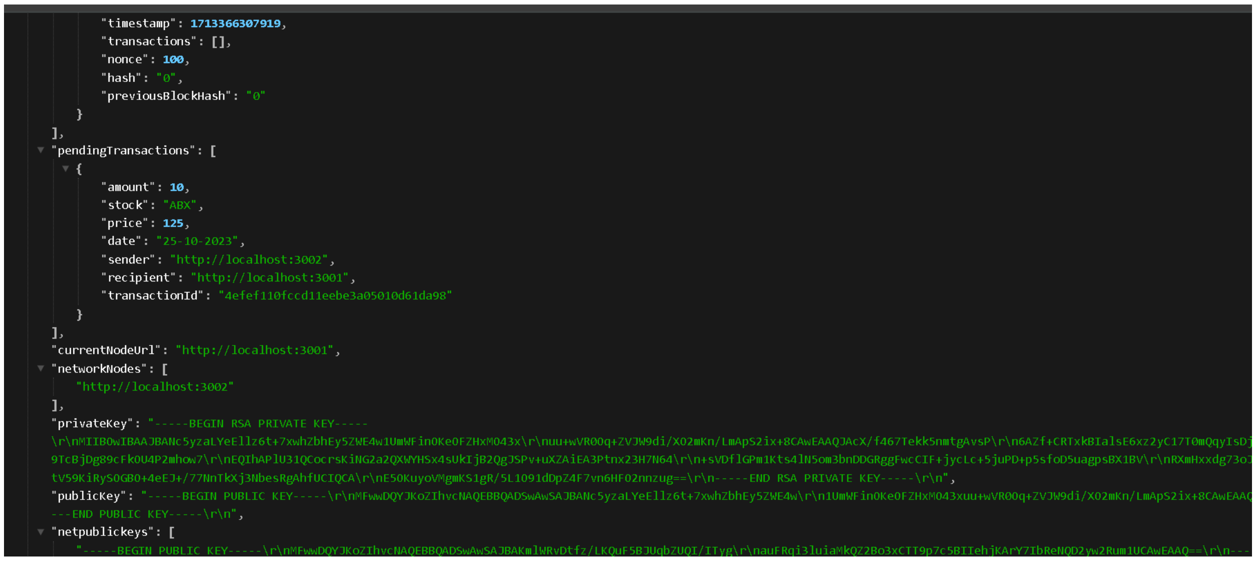
Body: raw [✓ JSON]

```
{
    "amount": 10,
    "stock": "ABX",
    "price": 125,
    "date": "25-10-2023",
    "sender": "http://localhost:3002",
    "recipient": "http://localhost:3001"
}
```

```
{
    "note": "Transaction created and broadcasted successfully."
}
```

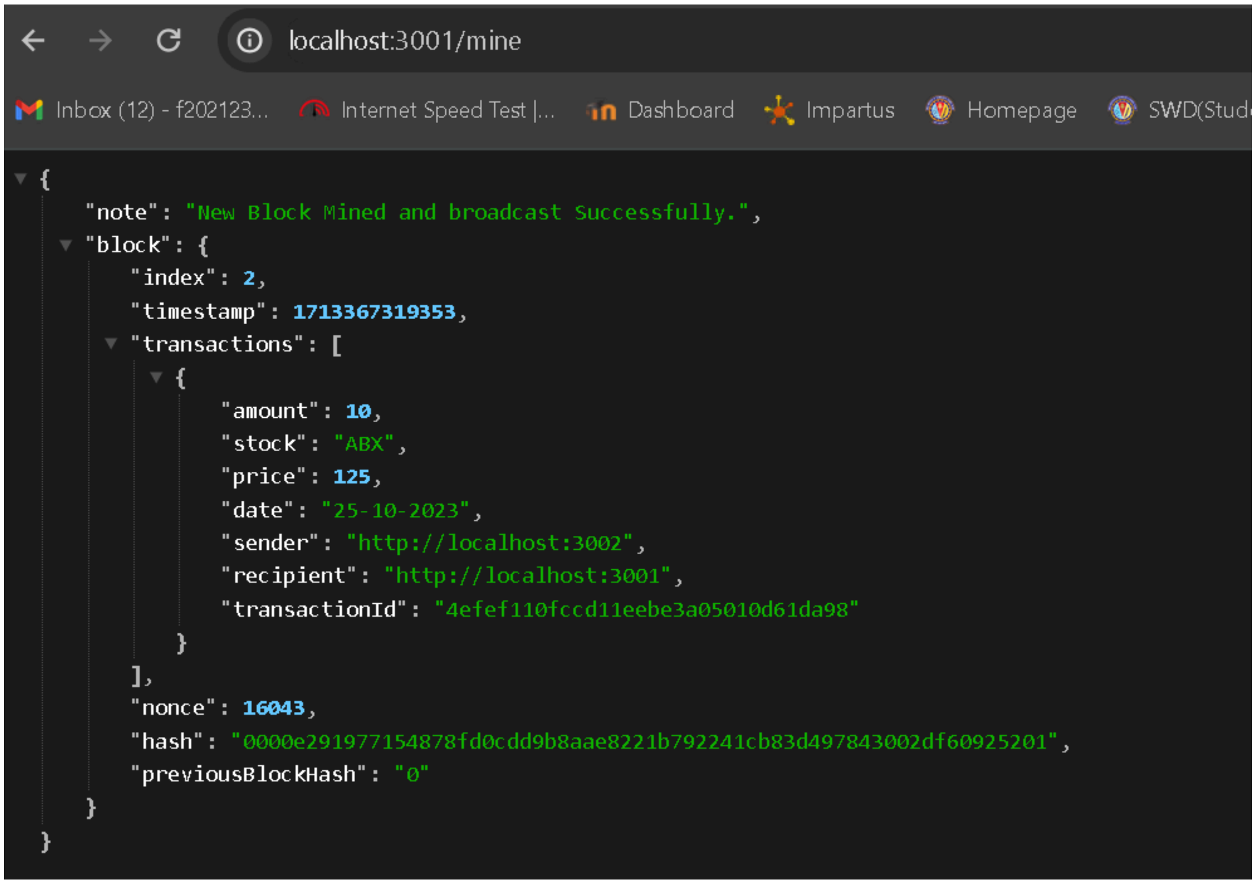
On successful verification and addition of transaction, you will get the response as mentioned in the pic above.

Now the user 1 looks like(showing the transaction part majorly):



```
            "timestamp": 1713366307919,
            "transactions": [],
            "nonce": 100,
            "hash": "0",
            "previousBlockHash": "0"
        }
    ],
    "pendingTransactions": [
        {
            "amount": 10,
            "stock": "ABX",
            "price": 125,
            "date": "25-10-2023",
            "sender": "http://localhost:3002",
            "recipient": "http://localhost:3001",
            "transactionId": "4efef110fccd11eebe3a05010d61da98"
        }
    ],
    "currentNodeUrl": "http://localhost:3001",
    "networkNodes": [
        "http://localhost:3002"
    ],
    "privateKey": "-----BEGIN RSA PRIVATE KEY-----
\r\nMIIBOwIBAAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w1UmWFin0Ke0FZHxM043x\r\nuu+wVR0Oq+ZVJW9di/X02mKn/LmApS2ix+8CAwEAAQJAcX/f467Tekk5nmtgAvsP\r\n6AZf+CRTxkBIalsE6xz2yC17T0mQqyIsDj
9TcBjDg89cFk0U4P2mhow7\r\nEQIhAPlU31QCocrsKiNG2a2QXWYHSx4sUkIjB2QgJSPv+uXZAiEA3Ptnx23H7N64\r\n+sVDflGPm1Kts4lN5om3bnDDGRggFwcCIF+jycLc+5juPD+p5sfoD5uagpsBX1BV\r\nRXmHxxdg73oJ
tV59KiRyS0GB0+4eEJ+/77NnTkXj3NbesRgAhfUCIQCA\r\nE50KuyoVMgmKS1gR/5L1091dDpZ4F7vn6HF02nnzug==\r\n-----END RSA PRIVATE KEY-----\r\n",
    "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBANc5yzaLYeEllz6t+7xwhZbhEy5ZWE4w\r\n1UmWFin0Ke0FZHxM043xuu+wVR0Oq+ZVJW9di/X02mKn/LmApS2ix+8CAwEAAQ
---END PUBLIC KEY-----\r\n",
    "netpublickeys": [
        "-----BEGIN PUBLIC KEY-----\r\nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKmlWRvDtfz/LKQuF5BJUqbZUQI/ITyg\r\nauFRqi3luiaMkQZ2Bo3xCTT9p7c5BIIehjKArY7IbReNQD2yw2Rum1UCAwEAAQ==\r\n---
```

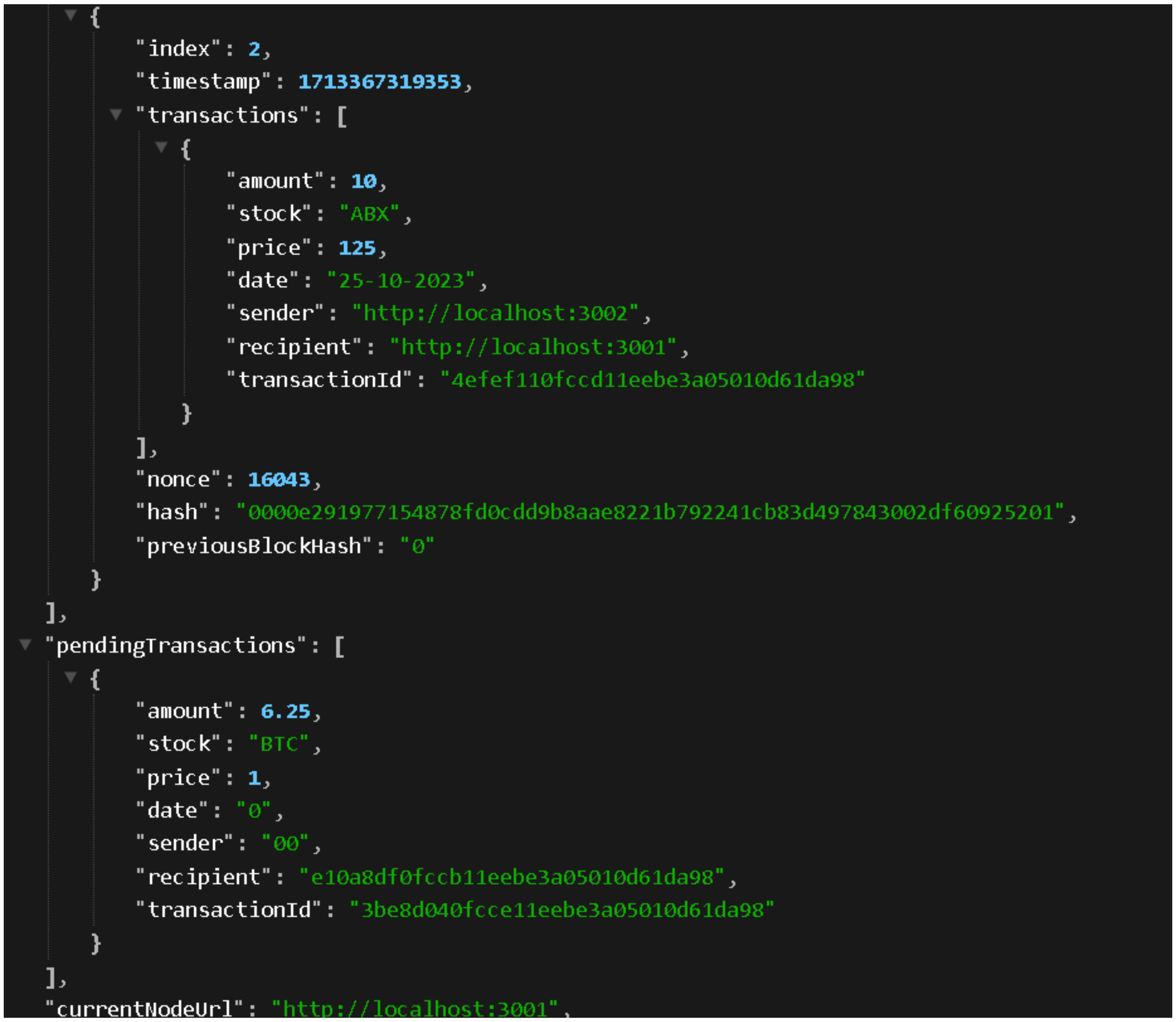
That transaction is added to a pendingTransactions array, now we need to mine a block to add this transaction to an array, so we head over to localhost:3001/mine (the /mine endpoint). On successful mining the mine will look like:



```
{
    "note": "New Block Mined and broadcast Successfully.",
    "block": {
        "index": 2,
        "timestamp": 1713367319353,
        "transactions": [
            {
                "amount": 10,
                "stock": "ABX",
                "price": 125,
                "date": "25-10-2023",
                "sender": "http://localhost:3002",
                "recipient": "http://localhost:3001",
                "transactionId": "4efef110fccd11eebe3a05010d61da98"
            }
        ],
        "nonce": 16043,
        "hash": "0000e291977154878fd0cdd9b8aae8221b792241cb83d497843002df60925201",
        "previousBlockHash": "0"
    }
}
```

The proof of work for our miner involves finding a hash that starts with ‘0000’, which is a majorly used in BITCOIN and we have used the same. The hash produced for the block starts with ‘0000’ as you can see above and hence proof of work is valid. The reward for the miner is added to pendingTransactions as a transaction and will be recognized in the blockchain when a new block is mined(as done in BITCOIN).

Now user1 looks like:



```
    {
        "index": 2,
        "timestamp": 1713367319353,
        "transactions": [
            {
                "amount": 10,
                "stock": "ABX",
                "price": 125,
                "date": "25-10-2023",
                "sender": "http://localhost:3002",
                "recipient": "http://localhost:3001",
                "transactionId": "4efef110fccd11eebe3a05010d61da98"
            }
        ],
        "nonce": 16043,
        "hash": "0000e291977154878fd0cdd9b8aae8221b792241cb83d497843002df60925201",
        "previousBlockHash": "0"
    }
],
"pendingTransactions": [
    {
        "amount": 6.25,
        "stock": "BTC",
        "price": 1,
        "date": "0",
        "sender": "00",
        "recipient": "e10a8df0fccb11eebe3a05010d61da98",
        "transactionId": "3be8d040fcce11eebe3a05010d61da98"
    }
],
"currentNodeUrl": "http://localhost:3001",
```

The mined block was added to the blockchain at the end of the last block, that is , it is added at index 2 next to the genesis block. Similarly, you can check all other functionalities that are included as endpoints in the api.js file. The function of all such endpoints are:

- '/prevBlock' : retrieves the last block of the blockchain
- '/uptodate' : if you add a new node(user) at a later stage in the blockchain, then after registering this node in the network this endpoint will take the longest existing copy of a blockchain out of every user and paste it to its own blockchain. Here we follow the principle that the longest existing blockchain is the true blockchain as it has done the most work(computation while

mining blocks). When you refresh the blockchain endpoint of the new user, you will see the longest copy of the blockchain on it and you can continue.

- '/block/:blockHash' : if you want to see a particular block from the blockchain, just put that block's hash in the :blockHash portion.
- '/transaction/:transactionId' : similar as above, put the transactionId to view a particular transaction and the block where it is stored.
- '/address/:address' : put the port(3001 for node_1) of a user at the :address place to view all transactions in which the user is involved and the users' account balance after all the trades(transaction). If you want to view the miner, just put the recipient address of any mining reward transaction in the :address space and you can view how many bitcoins the miner has collected as a reward till now.
- '/stocktransaction/:stock' : put the stock name at the :stock space to view all trades of that particular stock.
- '/day/:date' : put a valid date in the :date space to view all transactions(all stock trades) that happened on that day.