Pre-Demo

Background

In the Pre-Demo file, you can see all the necessary preparations for the blockchain system.

During the demo, a series of operations are performed on the blockchain system (I will expand on this later), and then a transaction of transferring coins from person A to person B takes place.

Since I recorded the demo on my PC, so that we could see that the money transfer and communication operations were working properly, I simulated 3 users from my PC.

Explanation

Part 1 (00:00 – 00:22)
At the beginning of the video, you can see the source code under the Blockchain.py file, while at the end of the file you can see the highlighted port address (Address 5000).

Then, it can be seen that in the console we paste the run command of the blockchain system.
You can immediately see that the server is in the air under port 5000 (as we would expect).
At any given moment you can stop running the server by pressing Ctrl + C.

Part 2 (00:22 – 00:36)
In this part of the video, you can see the nodes.json file: it contains the 3 users that will be active throughout the simulation (as I explained earlier). Each user has a unique port address (5001, 5002, 5003).

Part 3 (00:36 – 00:45)
In this part of the video, you can see the transaction.json file: it contains all the data that is relevant for us when we want to make a transaction via the blockchain system – sender, receiver, amount.

Part 4 (00:45 – end)
In this part of the video, you can see 3 different files: UserA.py, UserB.py, UserC.py that will simulate each of the 3 people in our blockchain system.

Each of the files contains the same source code we saw earlier (Blockchain.py), with the only difference being that we changed their names inside the code in line 174 (according to the file name) and updated their port number respectively (5001, 5002, 5003) in line 291.

Demo

Explanation

Part 1 (00:00 – 00:39)
At the beginning of the video, you can see the 3 files we prepared during the Pre-Demo.
For each such file, a new console is opened, that will demonstrate these 3 users 'sitting' on remote computers.
It can be seen that indeed for each console of each user, there is another port for the same user.

Part 2 (00:39 – 01:24)
In this point in the video, you can see the Postman software that will allow us to keep track of various requests and calls made on top of the blockchain.
It is important to note that here too, each port has its own window so we can simulate tracking for each of the different users.

The first command we run is a GET command called get_chain.
It can be seen that for each of the users, we get the desired output, with all the relevant data representing the "ancestor of the blockchain", i.e. the genesis block. For this point in time, each user's timestamp is different - since we have not yet merged the system.

Part 3 (01:24 – 03:30)
In this part of the video, we connect all the users to one system. As mentioned, currently each user is "in his own system" – while we want to link them to be one entity under the blockchain.

The second command we run is a POST command called connect_node.
I will explain from the point of view of UserA in port 5001.
You can see that we change the type of request to POST and copy from the nodes.json file the content we prepared in the demo preparation stage. Because we are simulating UserA, we will delete his port from the json file.

After sending the command, we can see that we have received a success message:
"All the nodes are now connected!", and a list of the other 2 users on the network.
We will perform the same operation for user B and C, when at the end of this step we get a fully connected graph – which is exactly what we wanted to get!

Part 4 (03:30 – 05:29)
At this point, we are performing a block mining operation.
The third command we run is a GET command called mine_block.
You can see that after running the command from UserA, we get a success message, which contains the 3 details we expected to receive:

- Amount: 1 ➔ we set the default amount for mine_block command to be 1
- Receiver: UserA ➔ we executed this command from UserA
- Sender: "20…" ➔ the address of the genesis block

The fourth command we run is a GET command called get_chain.
It can be seen that indeed the information has been updated and the new transaction has added to the blockchain system as we expected it to happen.
Then, we run the same get_chain command from UserB and UserC, and you can see that the information was not updated with them (as we might expect it to happen, because we connected them to a click graph).

To take care of this synchronization, we will need to take an active action to synchronize the blockchain.
The fifth command we run is a GET command called replace_chain.
We run it from UserB and UserC - and then you can see that we get the desired output, and the whole blockchain is synchronized in all the records so far.

Part 5 (05:29 – end)

At this point, we simulate the transfer of funds from UserA to UserC.

The sixth command we run is a POST command called add_transaction.

From the transaction.json file we copy the sending format, and update the values accordingly:

- Amount: 100 ➔ the amount of coins that we want to send
- Receiver: UserC ➔ the name of the receiver
- Sender: UserA ➔ the name of the sender

You can see that we received a message of success as we expected.

Also, the information was added to the 3rd block (so far there were 2 blocks, and now this operation is saved on top of a new block in the blockchain – block number 3).

The seventh command we run is a GET command called mine_block.

After running the command, we can see that the transaction has been entered and documented as we would expect it to happen (all the details of the transaction).

Now, let's note that if we run the get_chain command from UserB or UserC – we will not receive the newly added information (the currency transaction operation).

As before, in order for us to get the updated information found on the blockchain, we will need to run the replace_chain command that will align our data with the various users. Now, if we run the get_chain command again we can see that the information is in all users as expected.

In this way, you can continue to make transactions from any user to any user and as often as we like.