CSCI 401 Deliverable #7
(Due:12/4/2022 11:59PM)
Project #4: Blockchain Transaction Graph Database and Visual Explorer
Stakeholder: Prof. Bhaskar Krishnamachari
Mentor: Anoushka Agrawal
Team: Weilong Fam, Qinyu Richard Ge, Daniel Wang, Jiaxi Yang

Source code - Submission uses 'd3' branch
https://github.com/ANRGUSC/blockchain-graph-etl

Note that the most recently updated branch is the 'd3' branch, not the 'main' branch. Both are equally important, our demo and submission uses the 'd3' branch. You are encouraged to run code on both branches and compare the implementation approach in terms of how we generate the graphJson data and use them as a reference for future implementations.

Installation & Setup instructions

- Export transactions.csv from https://github.com/blockchain-etl/ethereum-etl

```
> ethereumetl export_blocks_and_transactions --start-block 0 --end-block
500000 \
--blocks-output blocks.csv --transactions-output transactions.csv \
--provider-uri
https://mainnet.infura.io/v3/7aef3f0cd1f64408b163814b22cc643c
```

- Neo4j-AuraDB Database
  We are using Neo4j AuraDB to work on a shared instance on the cloud.

  You need to create an Aura DB Instance, obtain the credentials and share it with whoever else is working on the project. Note that since this is a free instance, they will pause it periodically due to inactivity, so if you notice for some reason your project is not connecting to the AuraDB Instance, you should go to the AuraDB dashboard and check the status. If it is paused, just click on the play button to resume it.

  Then you can head to your preferred Neo4j GUI and connect to the instance via the credentials you created. Below is the link for Neo4j browser, you may also choose to use the Desktop version. It works the same.
    - https://browser.neo4j.io/?connectURL=neo4j%2Bs%3A%2F%2Fneo4j%403eab0bd3.databases.neo4j.io%2F&cmd=guide&arg=auradb%2Fmovies&_ga=2.129121292.1277831781.1665784426-255380095.1665784426

Section A: Demo Commands

Commands to run the server demo

1. cd flask-server
2. source venv/bin/activate
3. python3 server.py

Note: In order to run frontend demo, the server side must be started and running first.

Optional - Commands to run frontend demo (*using d3 branch, not main branch)

1. cd client/public
2. python3 -m http.server 8080

API endpoints

- localhost:8000/load
  load transactions.csv hosted on Google Drive into neo4j database, use the link in _load in models.py
- localhost:8000/transactions
  return all transactions in graph JSON format
- localhost:8000/cypher
  take the Cypher user inputs in the form in POST request, return the Cypher result in graph JSON format, throw an exception if the Cypher input is invalid
- localhost:8000/graph
  return the graph of all transactions on the front end

Video Demo & Slide Deck

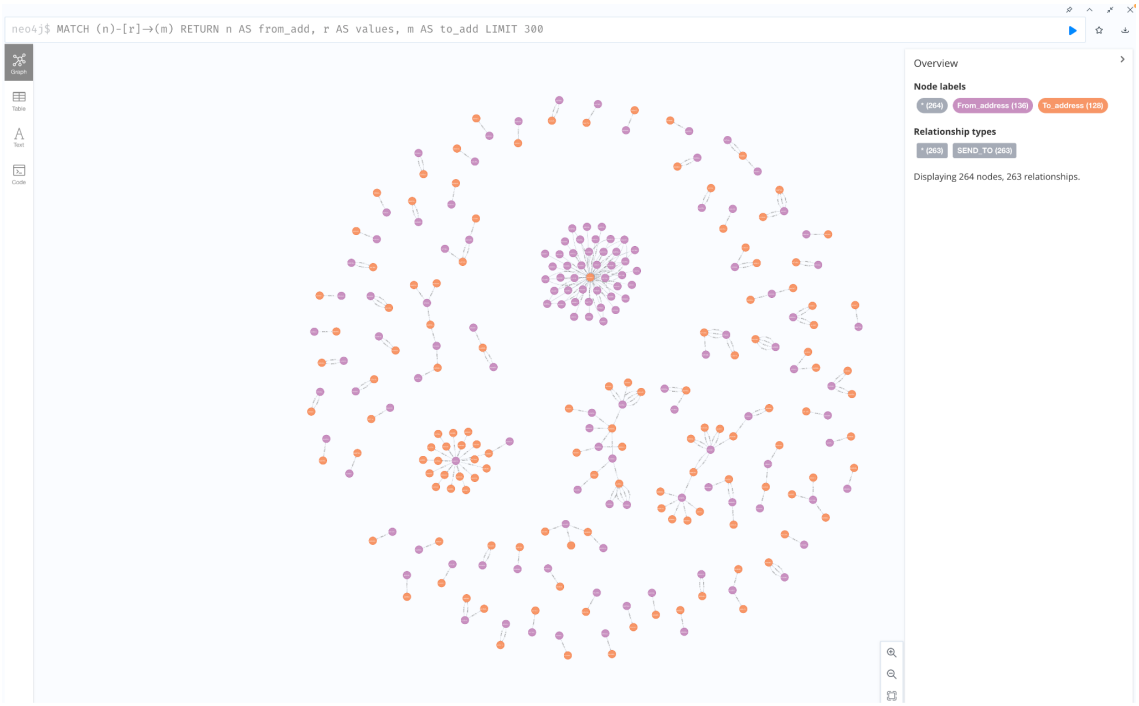
- Please see our upcoming video demo in our Final Project Presentation video about how to use this project

Section B. Neo4j Cypher Query & Data Model Documentation :

Cypher for loading transactions between from_address and to_address, and creating send_to relationships with the transaction value (LIMIT 300)

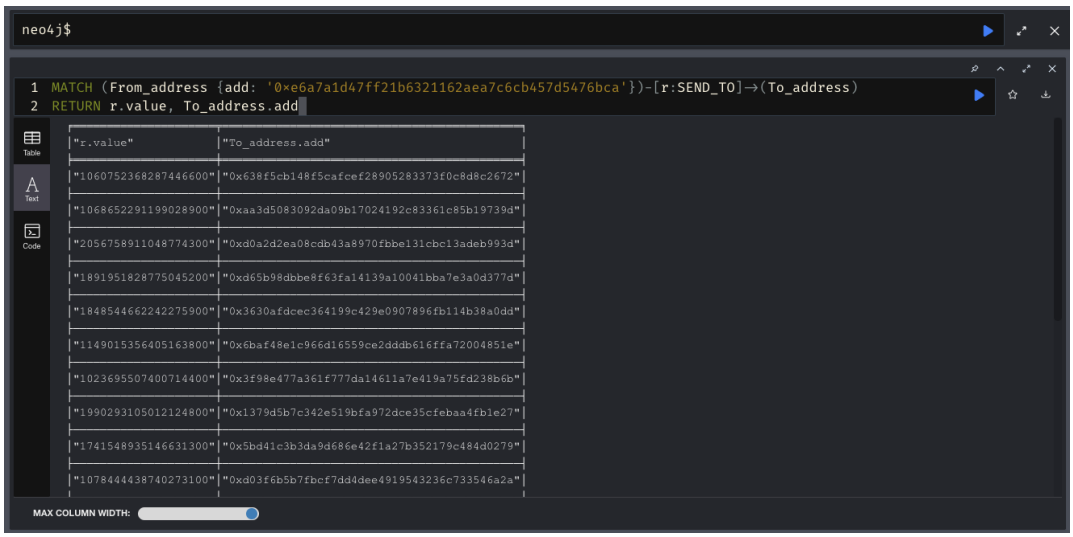
```
LOAD CSV WITH HEADERS FROM "file:///transactions.csv" AS csvLine
WITH csvLine LIMIT 300 WHERE csvLine.from_address IS NOT NULL AND
csvLine.to_address IS NOT NULL
MERGE (from_address:From_address {add: csvLine.from_address})
MERGE (to_address:To_address {add: csvLine.to_address })
MERGE (from_address)-[rel:SEND_TO {value: csvLine.value}]->(to_address)
```

This is also one of the first functions being executed in the backend - refer to _load method in models.py. As opposed to transactions.csv, we hosted a subset of this csv on google drive considering the original file was too large.



Cypher for subgraph: return all the transactions from a provided address

```
MATCH (From_address {add: '0x4d9279962029a8bd45639737e98b511eff074c21'})-[r:SEND_TO]->(To_address)
RETURN From_address as from_add, r as values, To_address as to_add
```



Section C: Code Documentations:

Root folder: project401

Backend: /flask-server

Runs on port 8000

Server.py (‘d3’ branch)

- Where the main backend source code resides
- Connects to the neo4j-auraDb instance via the credentials created
- Implementation for /cypher and /graph endpoints, both return graphJson data as a result. Please note that all implementations that queries the database likely use some kind of limit (300 records or 100 records) for demo purposes. You may edit this as you see fit.
- /graph endpoint essentially returns all nodes, relationships and their values in graphJson format.
- /cypher endpoint allows users to enter any query they want in the textarea shown in the GUI, however keep in mind that the user must have some prior knowledge or reference the data representation in the Neo4j database otherwise the query they choose to execute may result in errors. Refer to ‘Section B’ of this document or the load function in models.py for this.
    - Most importantly, we highly recommend that the return statement in the input cypher in the GUI should rename the 3 variables to ‘from_add’, ‘values’, and ‘to_add’ (Return ___ AS ____) as these are syntax that are hard coded in server.py code to generate the graph json data. You are of course free to change this.
    - For better understanding, here are some examples of input queries that work
        - All transactions
            - MATCH (n)-[r]->(m)
              RETURN n AS from_add, r AS values, m AS to_add
        - Transactions from a given address
            - MATCH (From_address {add: '0xa89ac93b23370472daac337e9afdf642543f3e57'})-[r:SEND_TO]->(To_address)
              RETURN From_address as from_add, r as values, To_address as to_add
    - Note the Return statement in both of the working input queries above. User’s input query should be similar.

- An example of how one transaction record is printed in the server side terminal is as follows.

```
--RECORD--
<Record from_add=<Node element_id='40' labels=frozenset({'From_address'}) properties={'add': '0xa89ac93b23370472daac3
37e9afdf642543f3e57'}> values=<Relationship element_id='191' nodes=(<Node element_id='40' labels=frozenset({'From_add
ress'}) properties={'add': '0xa89ac93b23370472daac337e9afdf642543f3e57'}>, <Node element_id='191' labels=frozenset({'
To_address'}) properties={'add': '0x92c27672fe65e002159ec2597fcf8897adbf5b29'}>) type='SEND_TO' properties={'value':
'8.60E+17'}> to_add=<Node element_id='191' labels=frozenset({'To_address'}) properties={'add': '0x92c27672fe65e002159
ec2597fcf8897adbf5b29'}>>
```

Models.py (‘d3’ branch)

- Implements the load method that is used to load csv hosted in google drive.
- Contains other endpoint methods for your reference and usage

Frontend: /client

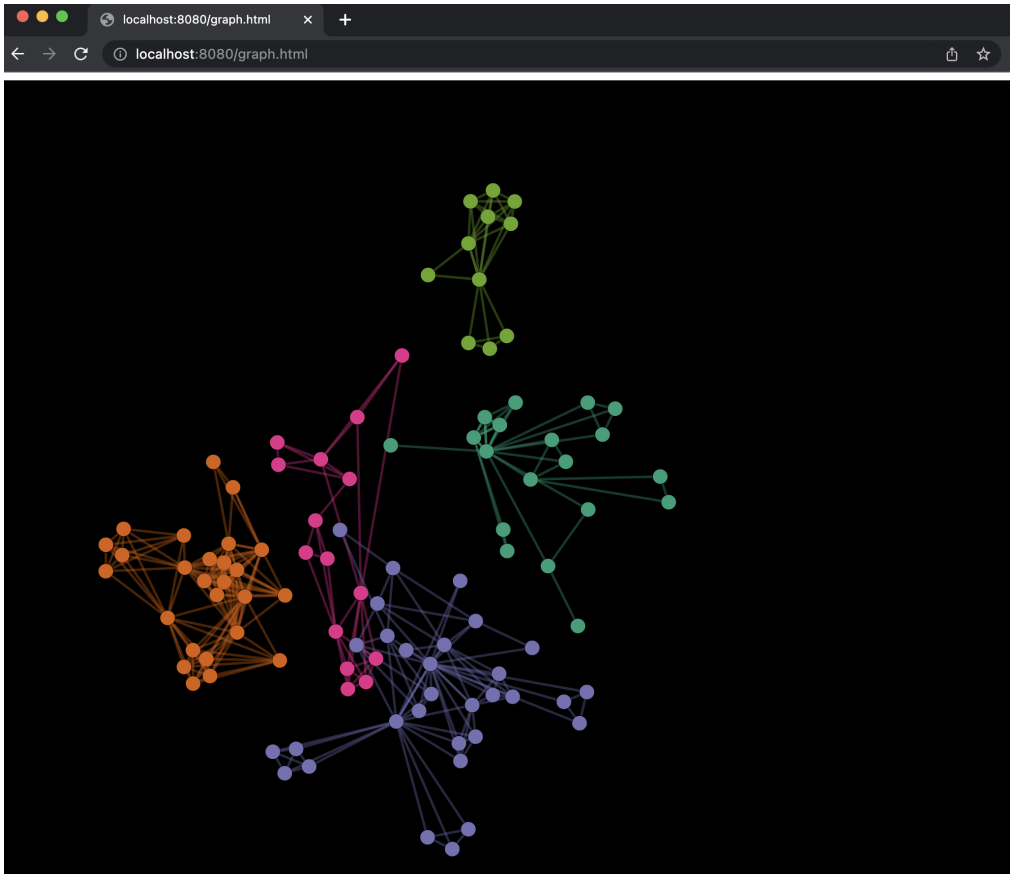
Runs on port 8080

Index.html (‘d3’ branch)

- Main frontend code
- d3.js framework implementation sample for /graph endpoint, feel free to refer to this section and tweak accordingly should you choose to use d3.js to visualize graphJson data returned from our endpoints. Note that there are still some bugs with this iteration as the multigraph is still not well-represented as opposed to Neo4j Dashboard visualization of the query result shown above. However, considering that the emphasis of our project this term is the api endpoints, we did not want to spend more effort in this regard as we wanted to work on the more important parts of the project. Furthermore, alchemy.js was our initial preferred choice, but due to various technical difficulties and hours of debugging, we had little choice besides pivoting to d3.js as a temporary solution. More info on Alchemy.js framework is below.

Secondary-Frontend (Unused): /neo4j-alchemy-cluster

- We’re able to utilize D3.js and GraphJSON to build our frontend graph framework
- More efficient handling than raw D3 JSON format.
- Benefit being we can use any custom dataSource defined
- Configuration based but can also be customized with javascript functions.
- Supports custom styling, highlighting, hover effects, clustering and interactive exploration.

- 

Graph shown by running on our own hosted server localhost:8080, displaying an html file with possible clustering graph we may implement via our custom GraphJSON files being output by our own cypher queries.

Section D: Future Implementations & Integrations (Graph JSON, D3.js, Alchemy.js)

- Add contract attributes to the node
    - We successfully added both boolean contract attributes is_erc20, is_erc721 to each node but we cannot get contracts.csv to get all the attributes values for these two contract variables
- Mock Cypher for ERC_token boolean node properties (contracts.csv unavailable)
    - ```
      LOAD CSV FROM "file:///demo.csv" AS csvLine
      MERGE (n:From_address {add:csvLine[0]})
      ON CREATE SET n.add=csvLine[0],n.is_erc20=csvLine[1], n.is_erc721 = csvLine[2]
      ON MATCH SET n.is_erc20=csvLine[1], n.is_erc721 = csvLine[2]
      ```

- Alchemy.js
    - Great alternative was our initial first choice as mentioned in Section C.
    - Excellent and robust library for future continuous integrations with graph networks without needing custom backend API endpoints.
    - Connect to a local running Neo4j database, run a cypher query to get the nodes, run a cypher query to get the relationships (edges).
    - Given more time, more work needs to be done to get the properly formatted .json file, which will in turn result in a nice web application visualization from your Neo4j database and is more customizable to your queries on the web application.

- Alchemy.js Documentation
    - Process begins with extracting data out of the Neo4j database.
    - You can build a cypher query by calling alchemy.plugins.neo4jBackend.buildQuery(input,queryType), where queryType is the key of the query template to use that is defined in the query key of the configuration.
    - This will return a cypher statement which you can then pass into alchemy.plugins.neo4jBackend.runQuery(cypherStatement [, callback]).
    - Passing in just a cypher statement and no callback will run your cypher query and update the alchemy graph with the results.
    - If we wish to modify/view the date before updating the graph, you can pass a callback to which receive the returned graph.JSON for us to work with.
    - This all will stop automatic graphUpdating, hence afterwards we will need to call the alchemy.plugins.neo4jBackend.updateGraph()

    - **Configuration**
        - Url to neo4j can be configured with the “url” key in the configuration file.
        - If it is running locally on some default port, then we do not have to configure anything explicitly.
      ```