Project Design Document

Group 3-5 (Group Set 3)

Project Background and introduction

The project at hand is the Blockchain Transaction Information Visualization System (BTIVS). This project focuses on presenting complex blockchain information to a user in a simple, understandable manner. This will allow the users to gain an understanding and further insight into the transaction patterns, network activity and smart contract interactions. The way that BTIVS will accomplish this is through visualising blockchain transaction data.

Team Introduction

Yiannis Kyritsis

Major: Internet of Things (IoT)

Project role: Leader

Preferred Contact Method: Email (103980370@student.swin.edu.au) or Discord (yiannis8655)

Hridoy Ahmed

Major: Data Science

Project role: Editor

Preferred Contact Method: Email (103798793@student.swin.edu.au) or Discord (.hridoyahmed)

Hiyoshi Woods

Major: Data Science

Project role: Researcher

Preferred Contact Method: Email (103392438@student.swin.edu.au) or Discord (.ppypy)

Project Requirement List and Description

For this project, five explicit core functional requirements must be adhered to in the development of this application/prototype to ensure that the project's aims are fulfilled.

These five requirements are:

1. Users of the Blockchain Transaction Information Visualization System (BTIVS) must be able to input a wallet address into a search bar on the website. This search should then display details of the wallet entered such as balance, and any other relevant data.
2. After searching for a specified wallet address, the web page will display a graph. This graph will display where each node represents a wallet address, and each edge will represent a transaction between the different connected addresses.

3. Users shall be able to interact with the graph in different ways, such as being able to explore the next or previous hop of a wallet address. Clicking on a node should expand details and allow the user to explore the paths of the transaction.
4. The transaction information shall be presented in a tabular format. This means it will be presented in a legible, clearly understandable table format.
5. All transaction data must be stored in a graph database. This will help with transaction data retrieval and visualisation.

Project Design

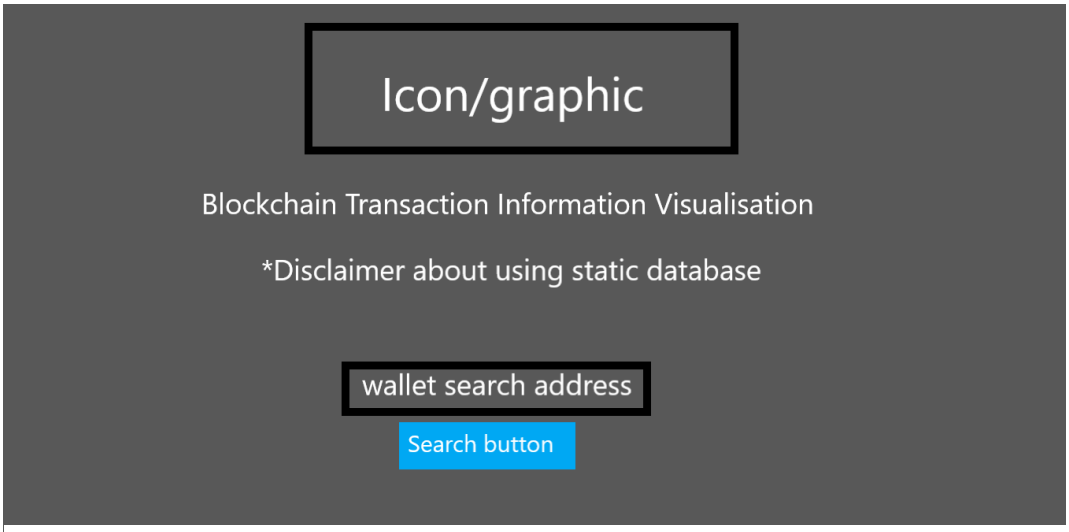


Figure 1

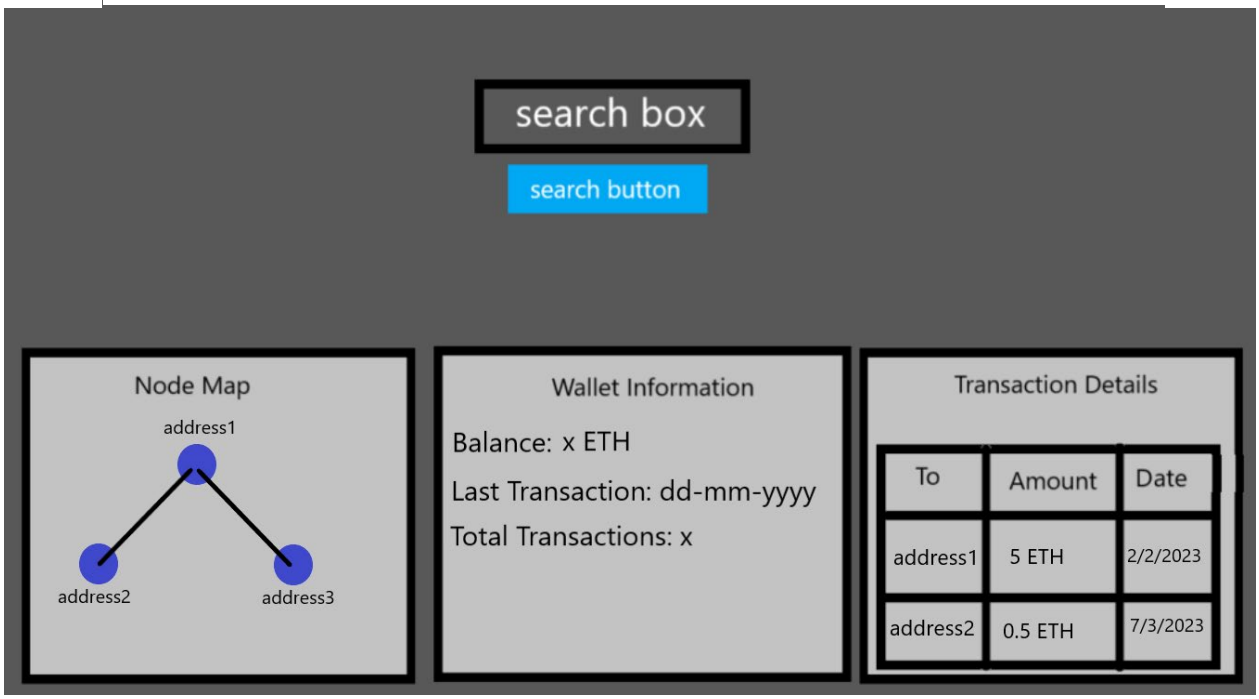


Figure 2

Prototype Explanation:

In Figure 1, you can see the home page. This page will contain an icon, the heading of the page and a search bar with a blue search button. The search bar is where the user will input their desired wallet address to lookup.

In Figure 2, this is the page that the user is presented with after they search for a wallet address. Firstly, they are presented with a clickable node map where they can view the different addresses that the user has sent or received balances from. This is identified by a blue dot with the name of the address below or above.

Next, the user is presented with the wallet information. This displays the balance, last transaction, and total transaction information to the user.

Lastly, their transaction details are displayed in a tabular format, where they can view where the wallet address has sent balances. They can also see the balance amount and the date that the transaction occurred.

Technical Explanation of the Static Website:

Firstly, the project is structured as a react application, with a separate 'Header' component for the header (Academind, 2021). As this is only a static website, we created and used a mock database to represent theoretical Ethereum wallet addresses and transactions. Each database entry contains properties such as:

1. Balance
2. Last transaction date
3. Total transactions
4. Connected addresses.
5. Transaction details

> We use two react state variables of 'address' and 'activeNode'. The address variable holds the address entered by the user for searching, and the activeNode variable stores information about the currently active or clicked address (Stack Overflow, 2021).

We created three functions 'handleSearch', 'handleKeyPress', and 'handleNodeClick(newAddress)' (W3Schools 2021).

The handleSearch function is activated when the user clicks the search button or presses enter, and searches for the address in our mock database.

The handleKeyPress function listens for the enter key press event to initiate a search.

The handleNodeClick is activated when a node on the node map is clicked and it sets the clicked address as the active node if it exists in the mock database.

In terms of UI, there is an input box where the user enters a wallet address and below it is a blue search button. When pressed, the user is presented with the node map, wallet details, and transaction table.

Our implementation of the node map is done by first creating an SVG element and using that as the canvas to draw the nodes and lines connecting them (MDN Web Docs, 2021). Essentially, the positions of the nodes are calculated using trigonometric functions to arrange them in an arc.

For each connected address, an angle is calculated using this formula.

$$\text{angle} = \frac{\pi}{\text{arr.length} + 1} \times (\text{index} + 1)$$

Next, we use a similar formula to calculate the coordinates of each node.

To draw a circle, we use the <circle> element, and we set the cx and cy attributes to the coordinates calculated previously. Then we use the <text> element to label each node with its respective wallet name/address. We then used the <line> element to connect the nodes and the coordinates of the circle to determine where the line should start and finish.

Clicking on a node triggers the handleNodeClick function which then sets the address as the active node as previously explained.

CSS styling is used throughout to make the web page nicer, and so that the wallet information is presented nicely and within boxes.

Backend Database Design:

Database: Neo4j Aura

Purpose: Neo4j Aura is a cloud-based graph database that allows the user (such as our group) to represent and query data in terms of entities and their relationships to each one. It was selected for this assignment because we can easily use an API such as Fast API to connect and do queries.

The data (relationships.csv and nodes.csv) that were provided as per the assignment were imported by using the Neo4j Aura import tool.

Schema:

Nodes: Represent wallet addresses, each node contains the following properties:

- 'addressId': A unique identifier for each wallet. It is the primary key/identifier for the nodes.
- 'type': Represents the type of wallet, being either a contract or eoa (externally owned address).
- 'total_in': Represents the sum of all incoming transactions to the wallet address. This was used for calculating the balance of the wallet.
- 'total_out': Represents the sum of all outgoing transaction from the wallet address. This was also used for calculating the balance of the wallet.

  Note: 'total_in' and 'total_out' were added post-import of the graph database. They were created so that we could calculate the balance of the wallet.

Relationships: Represent transactions between two addresses. The relationship properties are:

- 'from_address': Ethereum address of the sender.

- ‘to_address’: Ethereum address of the recipient.
- ‘hash’: Unique transaction hash.
- ‘Value’: Value of Ethereum transferred in the transaction.
- ‘input’: Input data for the transaction.
- ‘transaction_index’: Index of the transaction.
- ‘gas’: Gas limit set for the transaction.
- ‘gas_used’: Amount of gas used in the transaction.
- ‘gas_price’: Price of gas in Gwei.
- ‘transaction_fee’: Transaction fee.
- ‘block_number’: Block number in which the transaction was included.
- ‘block_hash’: Hash of the block.
- ‘block_timestamp’: Timestamp of when the block was mined.

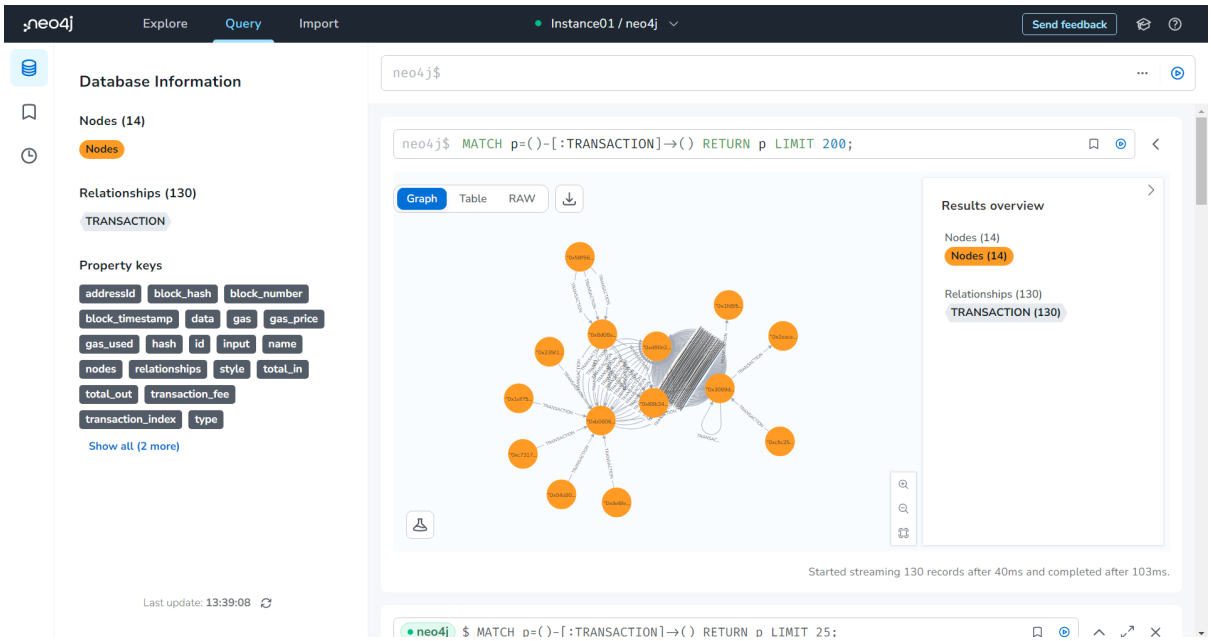


Figure 3 Neo4j Aura Database

API Design:

API Name and Description:

Wallet API: This API provides the ability to retrieve details, transactions, and connections related to Ethereum addresses.

Base URL:

http://127.0.0.1:8000

Endpoints:

/ (Root Endpoint):

URL: http://127.0.0.1:8000/

HTTP Method: GET

Description: A basic endpoint to check if the API is running.

Response Format: {"Hello": "World"}

One-Hop Connections:

URL: http://127.0.0.1:8000/one_hop/{address_id}

HTTP Method: GET

Description: Retrieves nodes (Ethereum addresses) that are one hop away from the provided address.

Request Parameters:

address_id: Ethereum address (e.g., 0xb0606f433496bf66338b8ad6b6d51fc4d84a44cd).

Response Format: JSON structure containing nodes and links representing connections.

Wallet Details:

URL: http://127.0.0.1:8000/wallet_details/{address_id}

HTTP Method: GET

Description: Fetches details of the specified Ethereum address.

Request Parameters:

address_id: Ethereum address (e.g., 0xb0606f433496bf66338b8ad6b6d51fc4d84a44cd).

Response Format: JSON structure containing the address ID, balance, block timestamp, and type of the address.

Transactions for Wallet:

URL: http://127.0.0.1:8000/transactions/{address_id}

HTTP Method: GET

Description: Retrieves all transactions associated with the specified Ethereum address.

Request Parameters:

address_id: Ethereum address (e.g., 0xb0606f433496bf66338b8ad6b6d51fc4d84a44cd).

Response Format: JSON structure containing a list of transactions. Each transaction contains hash, value, timestamp, gas, gas price, source address, and target address.

Function Description:

1. Functionality Name: Wallet Search

Purpose: The "Wallet Search" functionality provides users the ability to retrieve specific details about a particular Ethereum wallet. This includes the wallet's balance, and the type of address (e.g., contract, user).

Use Cases:

Step 1: Navigate to the Blockchain Transaction Information Visualisation (BTIVS) main page.

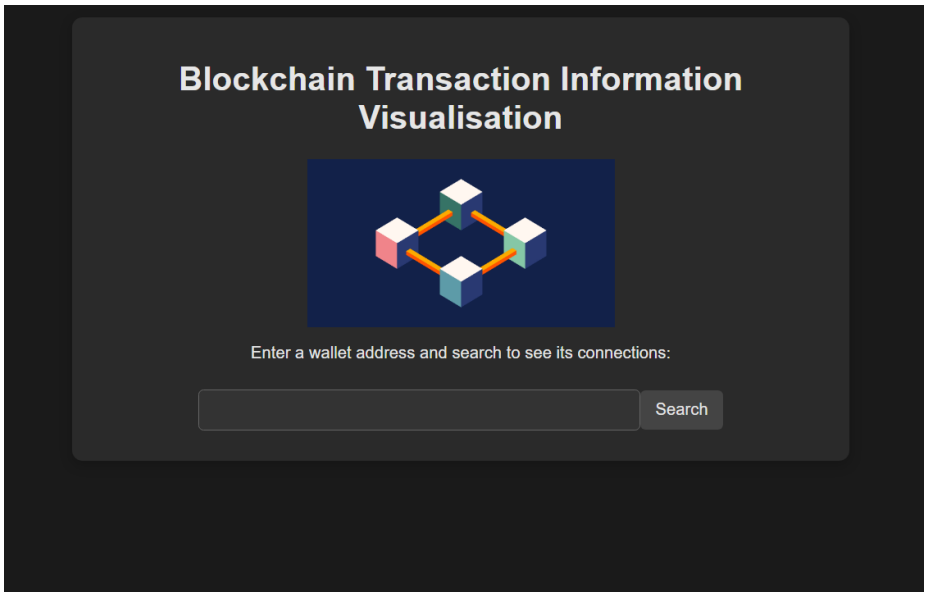


Figure 4 BTIVS Main page

Step 2: Enter the desired Ethereum wallet address into the provided input field and click on the "Search" button.

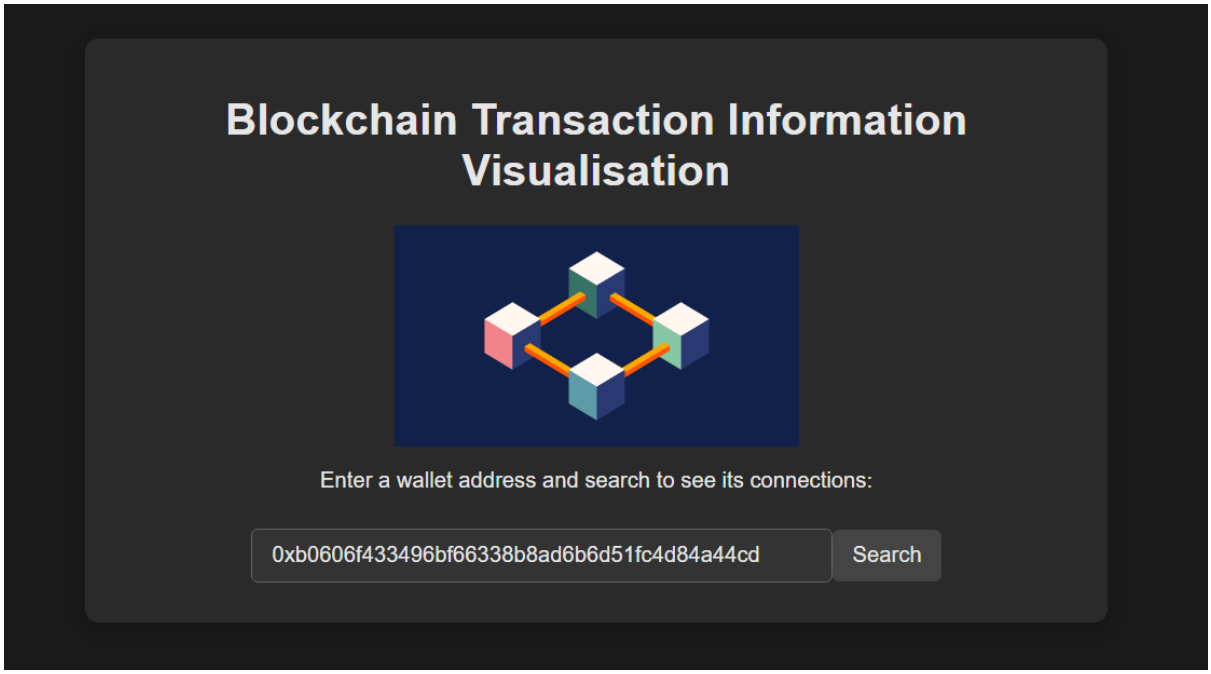


Figure 5 Search bar with wallet address

Step 3: View the wallet's detailed information displayed below the input field.

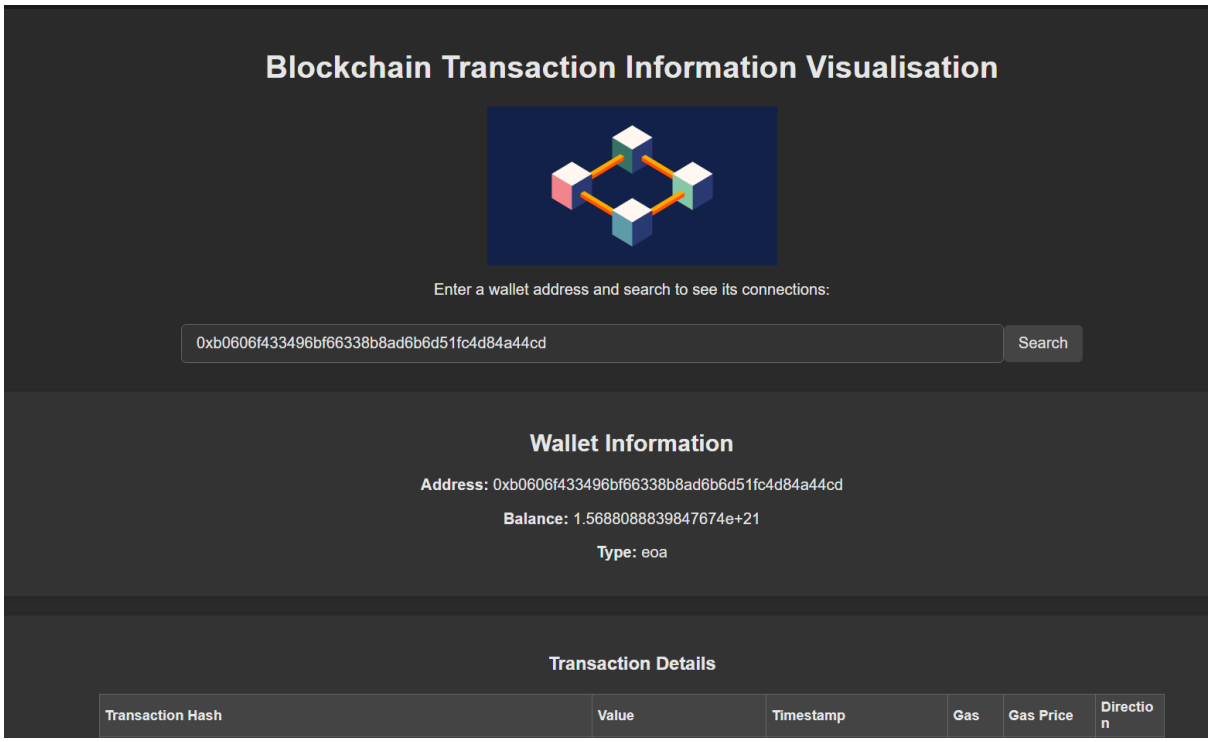


Figure 6 The page after searching for a wallet address

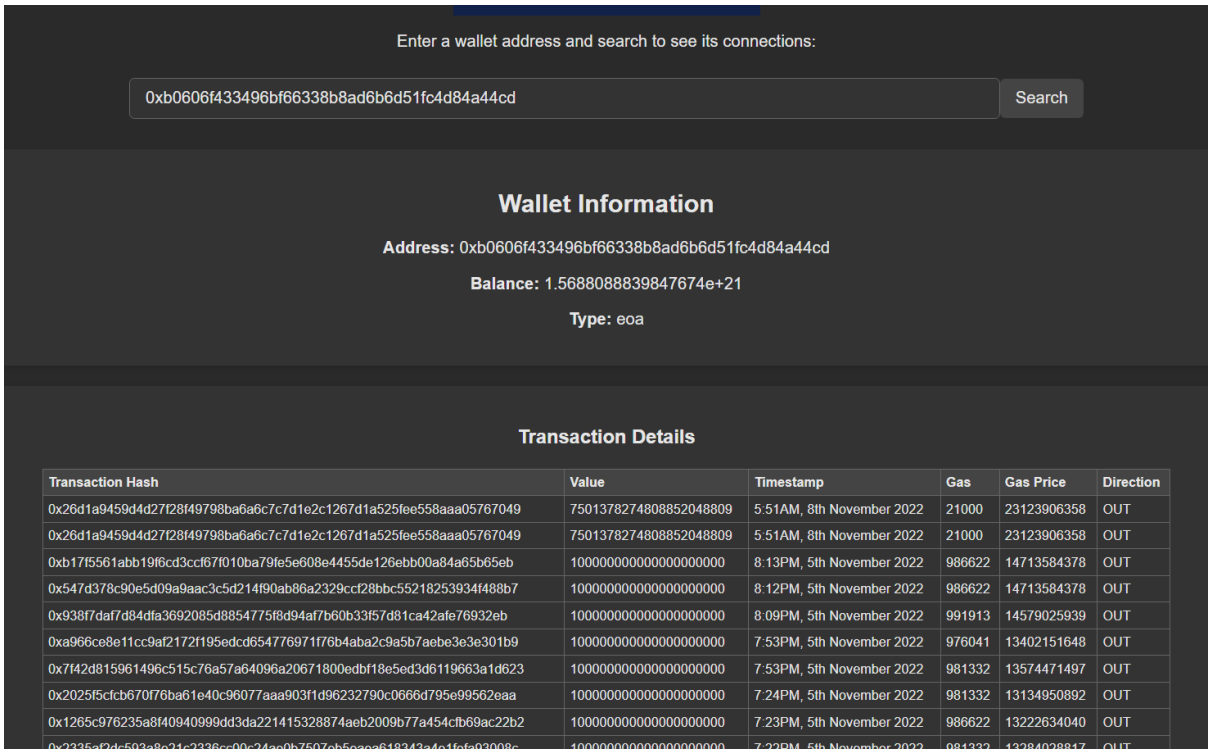
2. Functionality Name: Transaction Details Table

Purpose: This feature displays a comprehensive list of all transactions associated with a specified Ethereum wallet. This table showcases the transaction hash, value, timestamp, gas used, gas price, and whether the transaction was incoming or outgoing.

Use Cases:

Step 1: After performing a wallet search using the search bar, scroll down to view the transaction table.

Step 2: Examine the various transactions associated with the wallet. This includes reviewing transaction details and identifying patterns or specific transactions of interest.



Transaction Hash	Value	Timestamp	Gas	Gas Price	Direction
0x26d1a9459d4d27f28f49798ba6a6c7c7d1e2c1267d1a525fee558aaa05767049	7501378274808852048809	5:51AM, 8th November 2022	21000	23123906358	OUT
0x26d1a9459d4d27f28f49798ba6a6c7c7d1e2c1267d1a525fee558aaa05767049	7501378274808852048809	5:51AM, 8th November 2022	21000	23123906358	OUT
0xb17f5561abb19f6cd3ccf67f010ba79fe5e608e4455de126ebb00a84a65b65eb	100000000000000000000	8:13PM, 5th November 2022	986622	14713584378	OUT
0x547d378c90e5d09a9aac3c5d214f90ab86a2329ccf28bbc552182539334f488b7	100000000000000000000	8:12PM, 5th November 2022	986622	14713584378	OUT
0x938f7daf7d84dfa3692085d8854775f8d94af7b60b33f57d81ca42afe76932eb	100000000000000000000	8:09PM, 5th November 2022	991913	14579025939	OUT
0xa966ce8e11cc9af2172f195edcd654776971f76b4aba2c9a5b7aebe3e3e301b9	100000000000000000000	7:53PM, 5th November 2022	976041	13402151648	OUT
0x7f42d815961496c515c76a57a64096a20671800edbf18e5ed3d6119663a1d623	100000000000000000000	7:53PM, 5th November 2022	981332	13574471497	OUT
0x2025f5cfcb670f76ba61e40c96077aaa903f1d96232790c0666d795e99562eaa	100000000000000000000	7:24PM, 5th November 2022	981332	13134950892	OUT
0x1265c976235a8f40940999dd3da2214153288 74aeb2009b77a454cfb69ac22b2	100000000000000000000	7:23PM, 5th November 2022	986622	13222634040	OUT

Figure 7 Transaction Details

3. Functionality Name: Node Map

Purpose: The "Node Map" functionality provides a visual representation of Ethereum addresses connected to a specified wallet. The map displays nodes representing Ethereum addresses, and lines (or links) show the connections based on transactions. This visualization allows users to explore connections one hop at a time, offering an understanding of a wallet's transactions.

Use Cases:

Step 1: After performing a wallet search using the "Search for Wallet Details" functionality, navigate further down to the Node Map section.

Step 2: Observe the primary node representing the searched wallet address and its directly connected nodes.

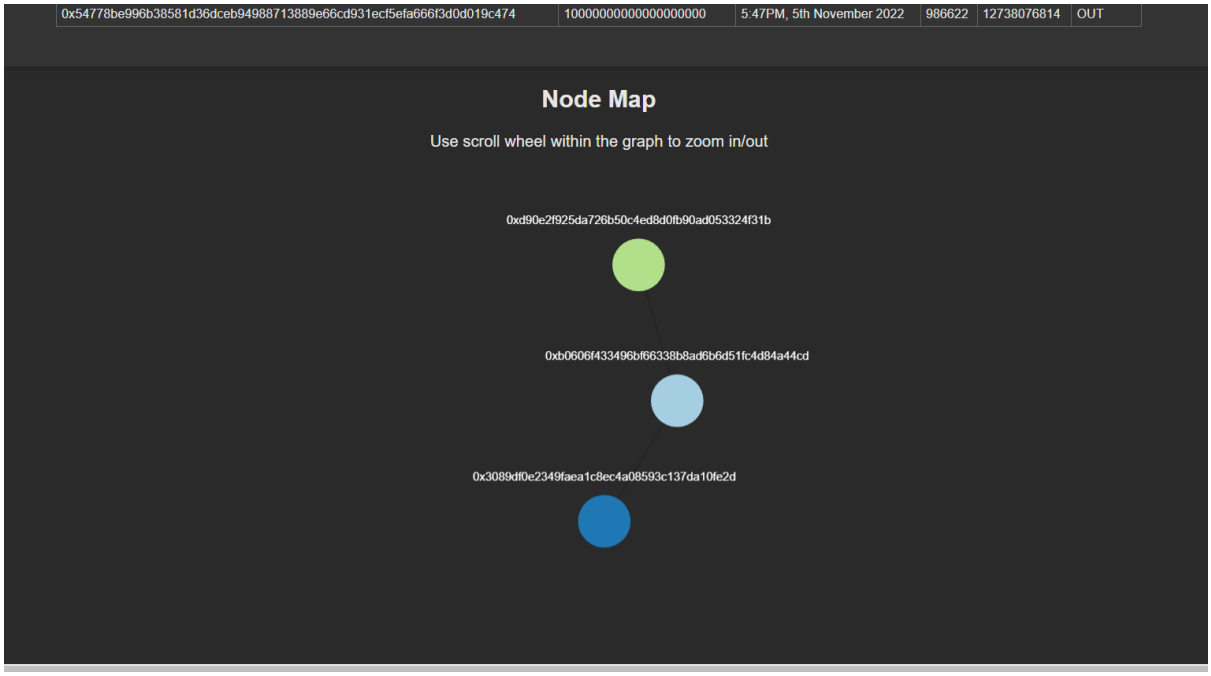


Figure 8 Node Map

Step 3: Click on any connected node to explore its connections further. This will recenter the map on the clicked address, allowing users to explore the network one hop at a time.

Project Deployment Instructions:

Project Deployment Instruction:

1. **Environment Setup:**

- Ensure you have Python 3.8 or above installed.
- Install Anaconda
- Create a new conda environment:
  - ‘conda create -n assignment python=3.8’
- Activate the environment:
  - ‘conda activate assignment’

2. **Dependency Installation:**

- With the conda environment activated, install essential Python packages for the backend:
  - ‘pip install fastapi neo4j uvicorn’

3. **Backend Execution:**

- Navigate to your project's root directory.
- Start the FastAPI server:
  - ‘uvicorn main:app –reload’

4. **Frontend Setup:**

- Ensure Node.js and npm are installed.
- Traverse to the frontend project directory.
- Install required node modules:

    - ‘npm install’
- Launch the frontend application:
    - ‘npm start’

5. Accessing the Application:

- Open a web browser.
- Navigate to http://localhost:3000 to interact with the application interface.

References:

W3Schools. 2021. "React Components". Available at: https://www.w3schools.com/react/react_components.asp [Accessed 26 August 2023].

Academind. 2021. "React.js & Node.js (MERN) Tutorial - Full ECommerce in 5 Hours [2021]". YouTube Tutorial. Available at: https://www.youtube.com/watch?v=0divhP3pEsg [Accessed 26 August 2021]

Stack Overflow. 2021. "How to handle state in React". Available at: https://stackoverflow.com/questions/57709403/how-to-handle-state-in-react [Accessed 26 August 2023].

MDN Web Docs. 2021. "SVG Tutorial". Available at: MDN Web Docs SVG [Accessed 26 August 2023].