Blockchain Transaction Information Visualization System

COS30049 – Computing Technology Innovation Project

Group 3 - 25

Nguyen Nam Tung 103181157
Xuan Dat Le 103487949
Abdullah Al Taskin 103793044

Table of Content

1. Project Background and Introduction

Blockchain, often described as a distributed ledger or a decentralized database, is maintained across numerous nodes in a computer network. While they've become synonymous with cryptocurrencies, ensuring secure and decentralized transaction records, their application extends well beyond the digital currency realm, impacting sectors from finance to healthcare (Hayes, 2023). However, as the reach of blockchain technology broadens, a significant challenge emerges: how to make the complex dynamics of blockchain transactions comprehensible to all, regardless of their technical knowledge.

The Blockchain Transaction Information Visualisation System is our group's response to this issue. Utilizing cutting-edge visualizing techniques and a user-friendly interface, we intend to transform complex blockchain data into simple visuals, allowing users to view and investigate blockchain transaction patterns and thus gain valuable insights.

2. Team Introduction

Team 3-25 is an enthusiastic group of individuals with backgrounds in AI and Data Analysis. Our team consists of three members: Xuan Dat Le, Nguyen Nam Tung, and Abdullah Al Taskin. Each member contributes a distinct set of skills and abilities.

Our team is led by Xuan Dat Le, an expert front-end programmer who has a lot of mastery in skills such as ReactJS and TailwindCSS. As the project's primary developer, Dat plays a crucial role in guiding the technical aspects of the project.

As our second developer, Nguyen Nam Tung possesses a wide range of database skills, particularly in areas such as database optimization and graph database. Tung's knowledge extends to visualization libraries such as D3 JavaScript and Sigma JavaScript, ensuring that the transaction data is effectively displayed and visualized.

Last but not least, Abdullah Al Taskin is the designer and communicator of our group. Abdullah ensures the effective communication in the team and he is also responsible for sketching the initial designs for our blockchain visualization system.

Together, we are committed to developing a user-friendly blockchain visualization website. This platform will enable users to extract valuable insights from blockchain transactional patterns, thereby developing their comprehension of this revolutionary technology.

3. Project Requirement List and Description

Our team has defined a list of website functionalities that must be included in order to meet the project requirements. This includes:

1. Searching Functionalities

Description: Users can input a wallet address into a displayed search field, which is located in the navigation bar on the top of the website.

Purpose: This allows users instant access to specific wallet information and its transaction history.

2. Wallet Data Display

   Description: Upon a successful search, relevant data related to the wallet address, such as its balance, is displayed.

   Purpose: This offers a concise overview of the present condition of the wallet and its record of transactions.

3. Transaction Graph

   Description: A network graph that visualizes the flow of transactions. Nodes represent wallet addresses, while the edges depict the transactions.

   Purpose: This visualization provides a view of the transaction flows between various wallets, aiding in pattern recognition.

4. Interactive Graph Features

   Description: : Allows users to click on nodes to dive deeper into transactional chains.

   Purpose: This feature enables users to explore and trace transaction pathways, understanding connections and patterns.

5. Transaction Table

   Description: This table is the transaction records which are displayed in a tabular format

   Purpose: The user can view and examine the details of the transaction data including information on the sender, receiver, time of transaction, and the amount involved.

6. Graph Database

   Description: All transaction data is stored in a graph database.

   Purpose: This feature facilitates efficient retrieval and representation of data

4. Project Design

4.1. Front-end prototypes

4.1.1. Design Sketches

Our website's design sketches have been created using Canva, a renowned online graphic design platform:

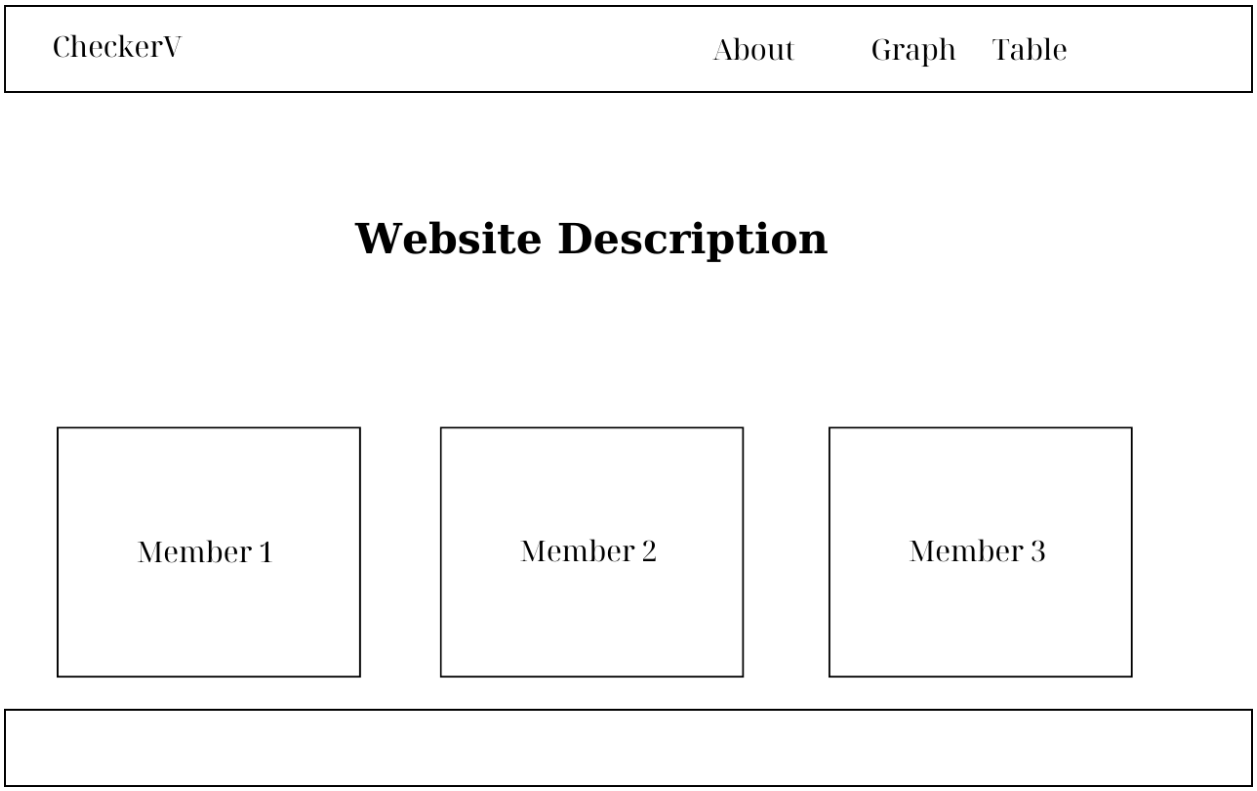


Figure 1: About page design sketch

This is our website's About page. It displays all project team members and their respective roles (see Figure 1). To ensure that our website is consistent and user-friendly, all of the pages will be linked in a singular navigation bar at the website's top. This facilitates simple navigation across the platform, ensuring that users can quickly locate the desired data or feature without unnecessary clicks or searches (redballoon.in, n.d.).

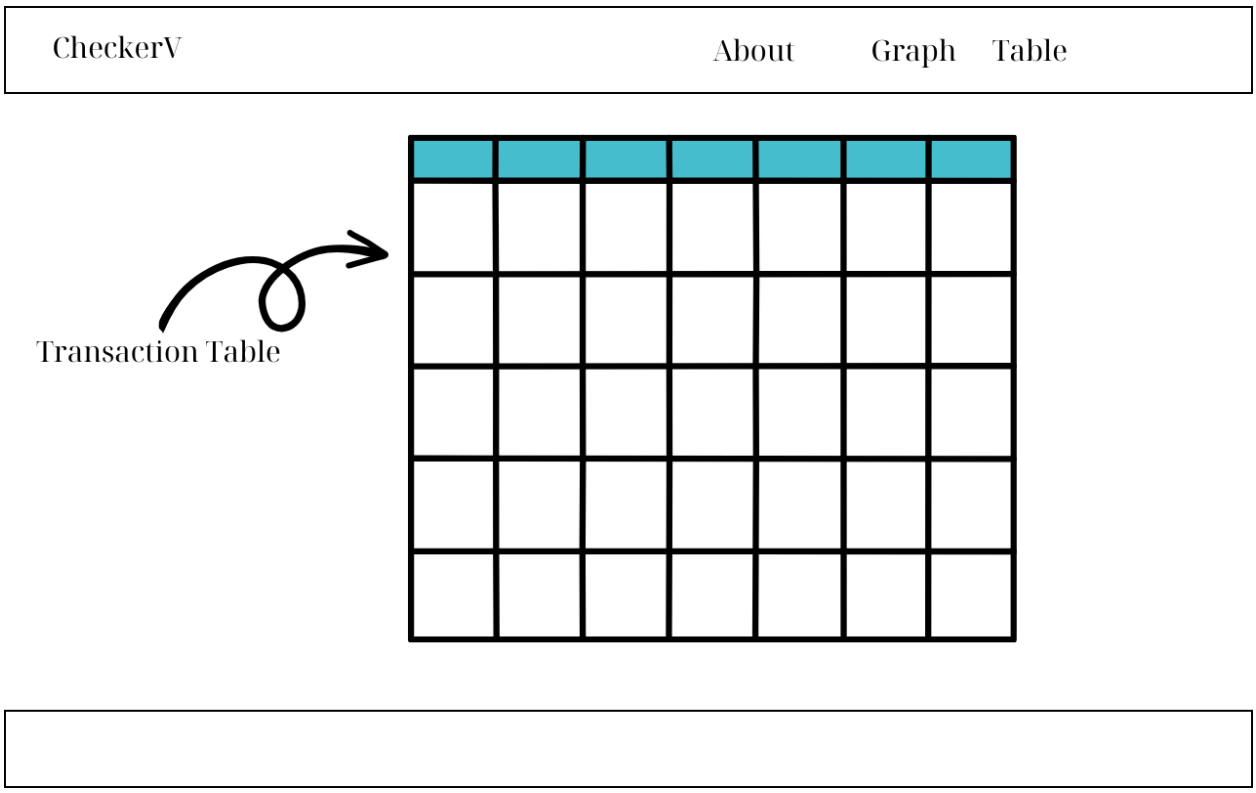


Figure 2: Transaction page sketch

All the transaction records will be shown in the Table page in tabular format (see Figure 2).

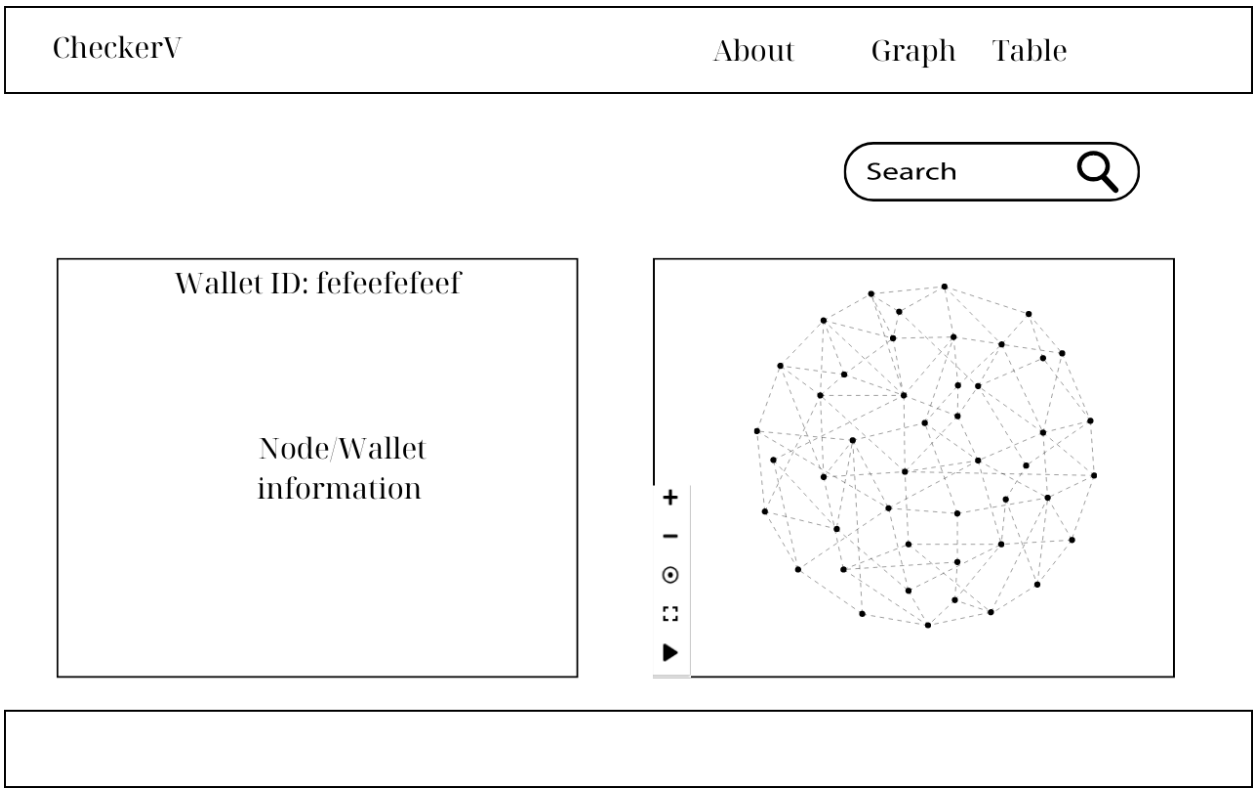


Figure 3: Main page sketch

Lastly, we display the wallet data and the transaction network graph in the main page of the website (see Figure 3). In terms of web page architecture, the sections such as the wallet information and the visualization are divided logically and effectively, allowing the user to navigate the page efficiently. For the graph visualization, users are not just presented with static data. Using a control panel at the bottom right of the visualization section, they can interact with the visualization by delving deeper into transaction paths, zooming, dragging the address nodes, and even expanding the view of the graph. This interaction enables users to customize their view to focus on what is essential for them, thereby enhancing the user experience. Last but not least, a search bar is displayed on the top of the visualization, enabling users to quickly enter a wallet address or other relevant criteria, narrowing down vast blockchain data sets to specific transactional information.

4.1.2. Final Design Drawing

Using Figma, a web-based interface design tool that enables designers to collaborate in real-time, we were able to create the final design drawing of our website based on the initial sketches (see Figure 4, Figure 5, Figure 6 and Figure 7):

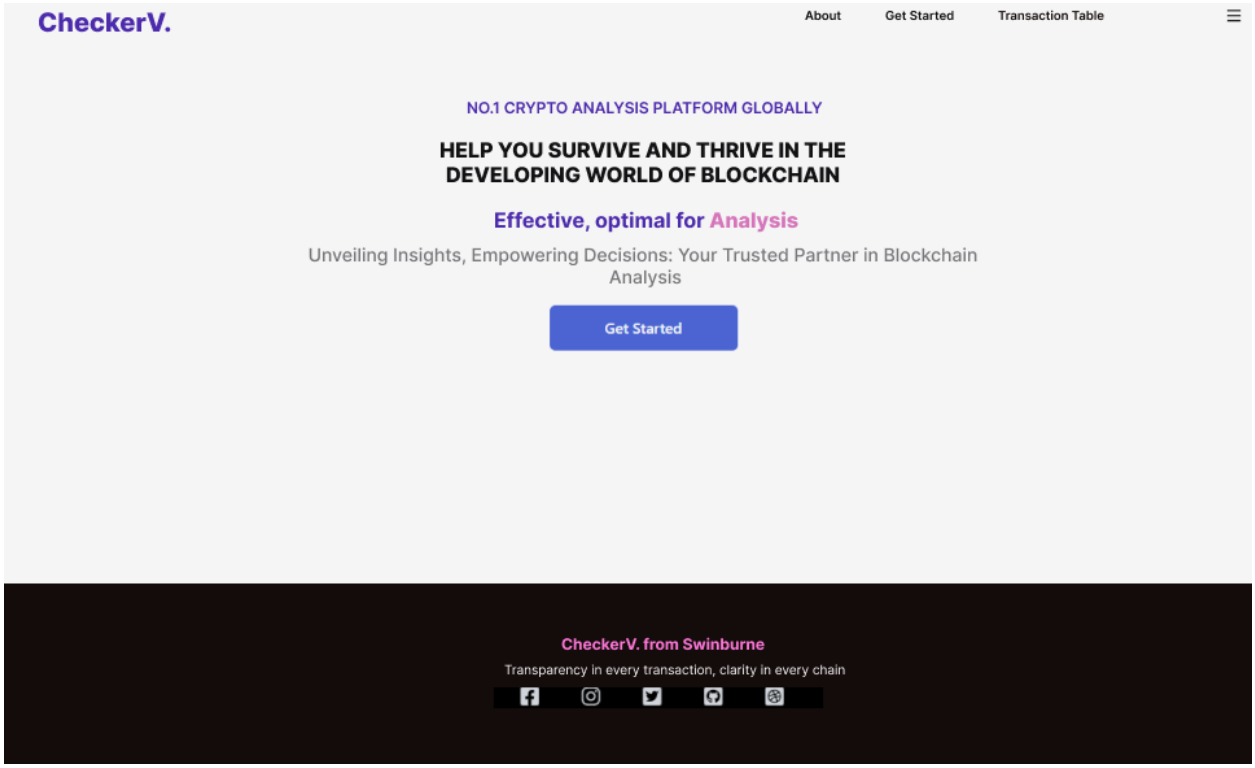


Figure 4: Final design drawing: Homepage page

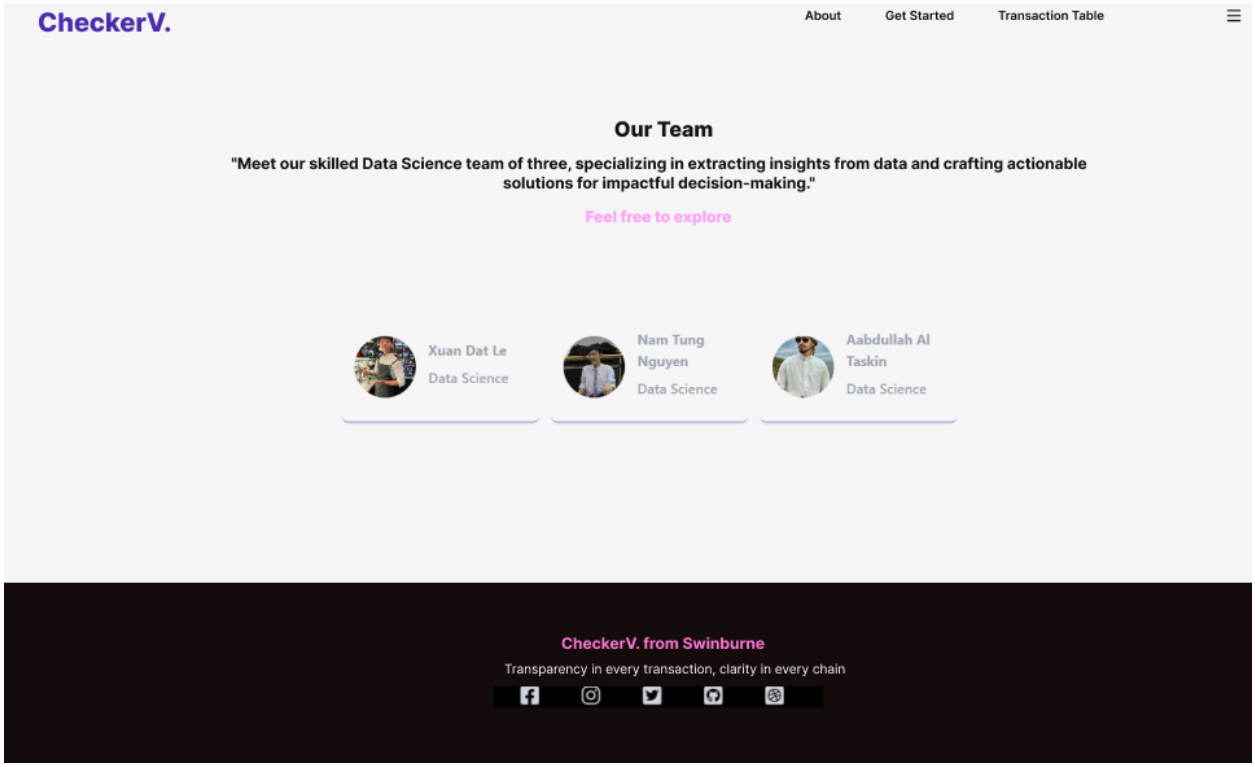


Figure 5: Final design drawing: About page

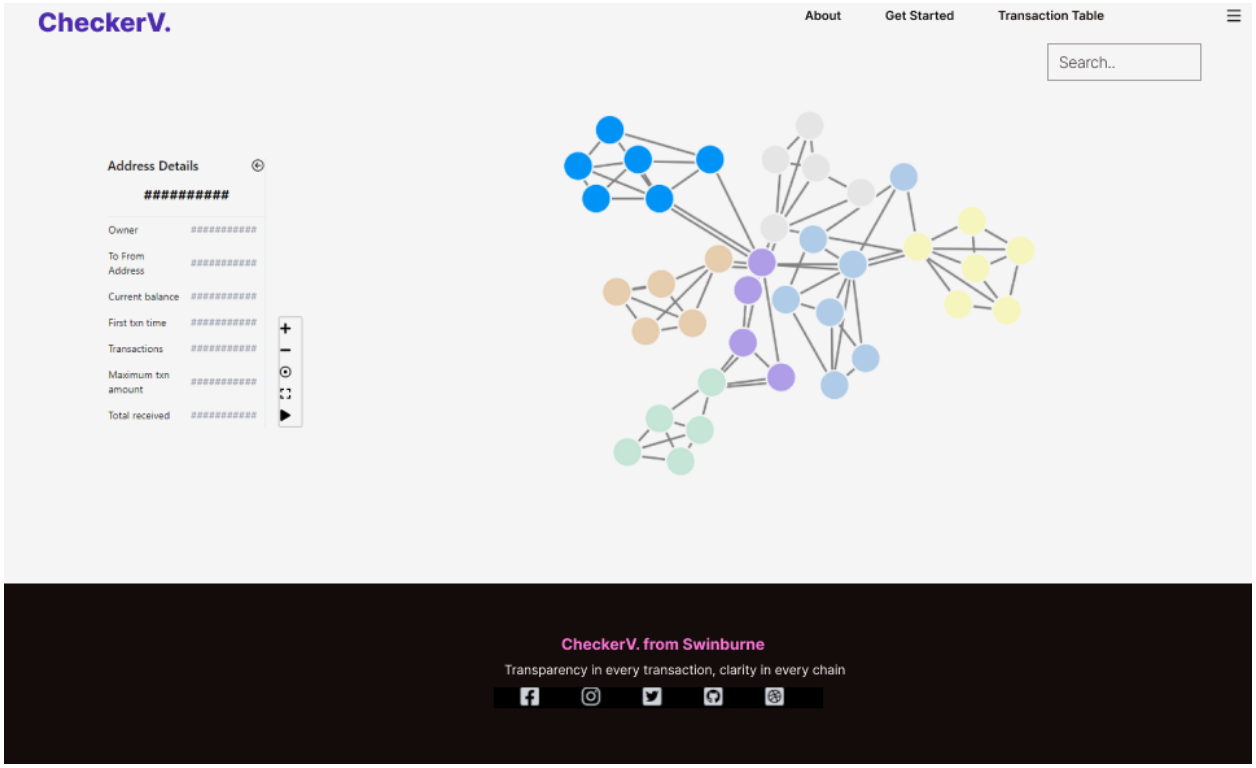


Figure 6: Final design drawing: Main page (Graph image source: https://linkurious.com/blog/why-graph-visualization-matters/)

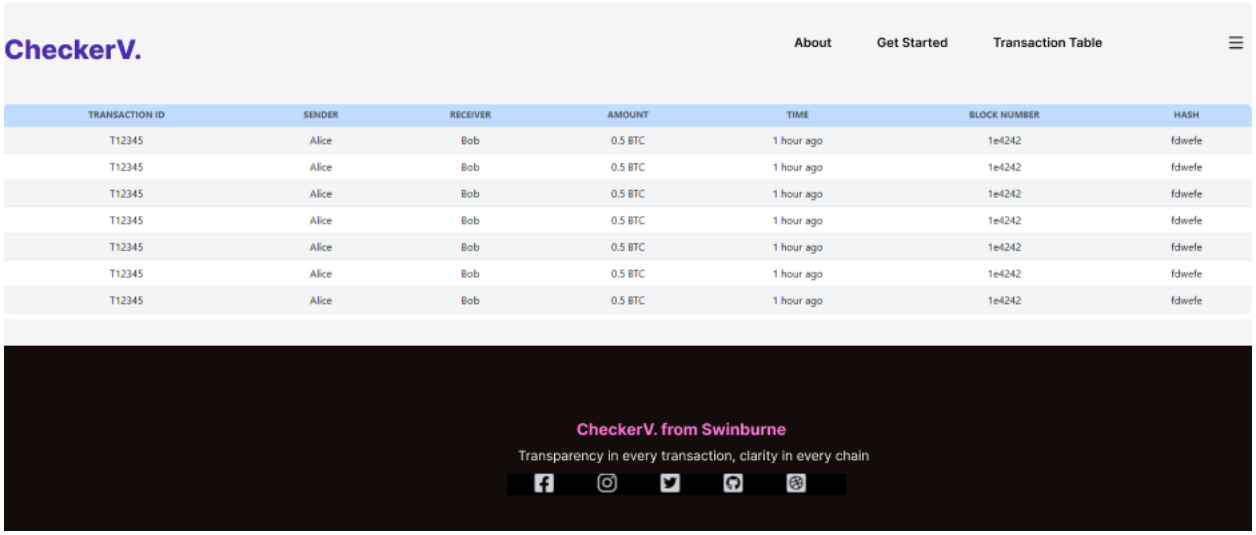


Figure 7: Final design drawing: Transaction page

We've adopted a clean, uncluttered design with high contrast text and background, ensuring all information is easily readable. Additionally, the use of whitespace and hierarchical design helps highlight important data, making it stand out and ensuring users can quickly identify key information (Soegaard, n.d.).

4.1.3. Final Website

Here is the final product for this assignment that we have developed based on the final design drawing: Blockchain Transaction Information System front-end website (see Figure 8, Figure 9, Figure 10 and Figure 11)

:

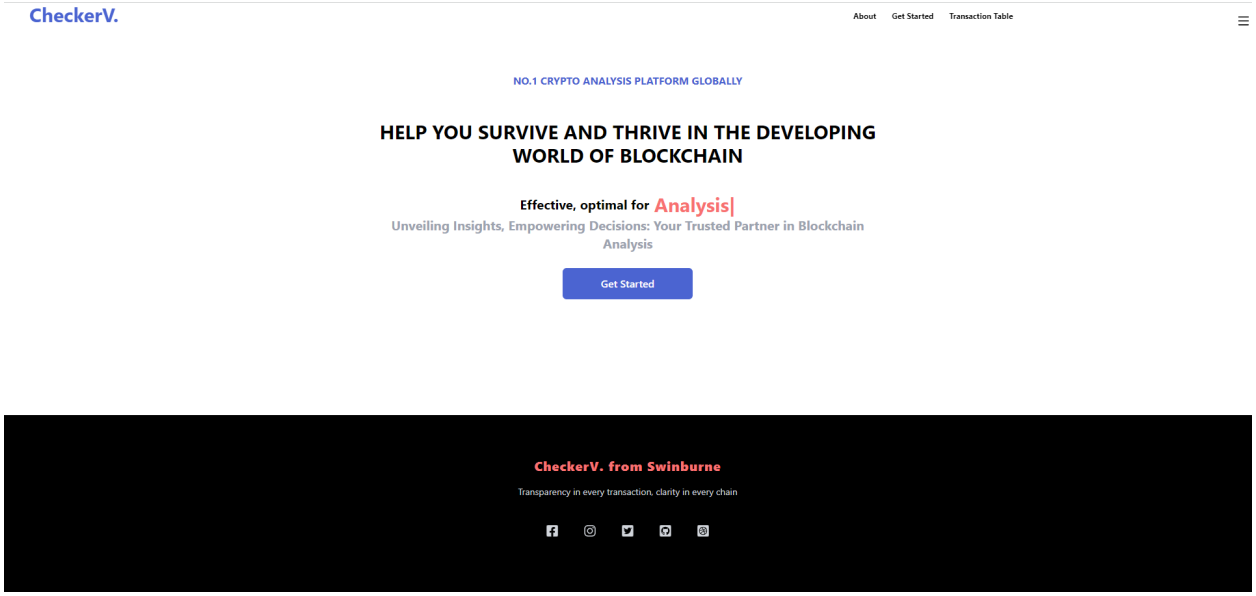


Figure 8: Main page

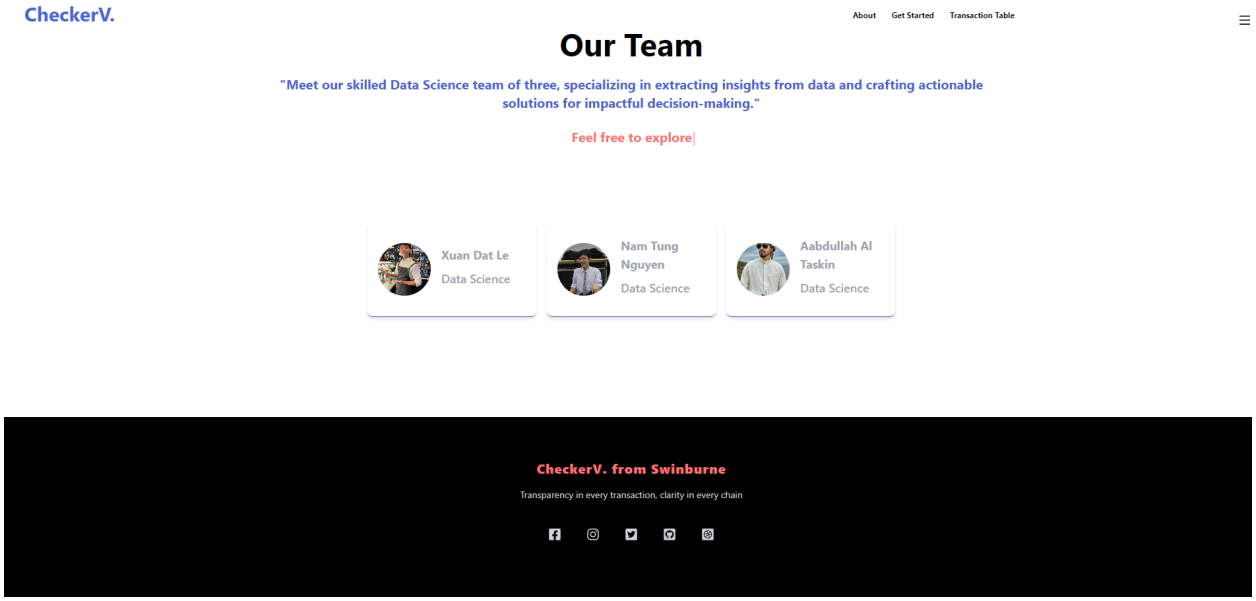


Figure 9: About page

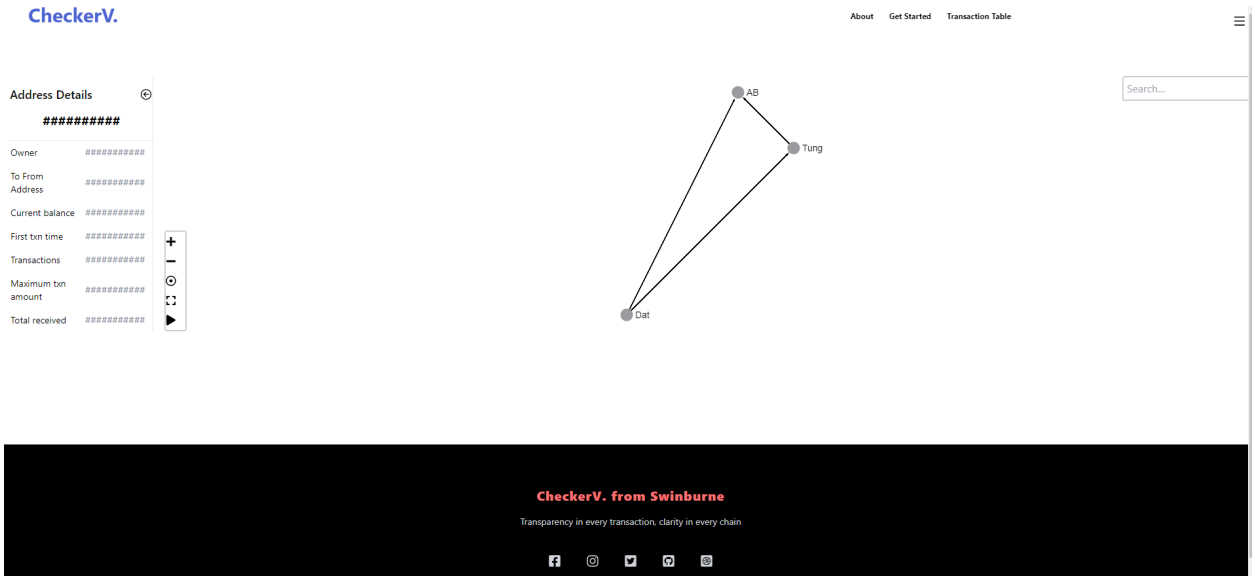


Figure 10: Main page

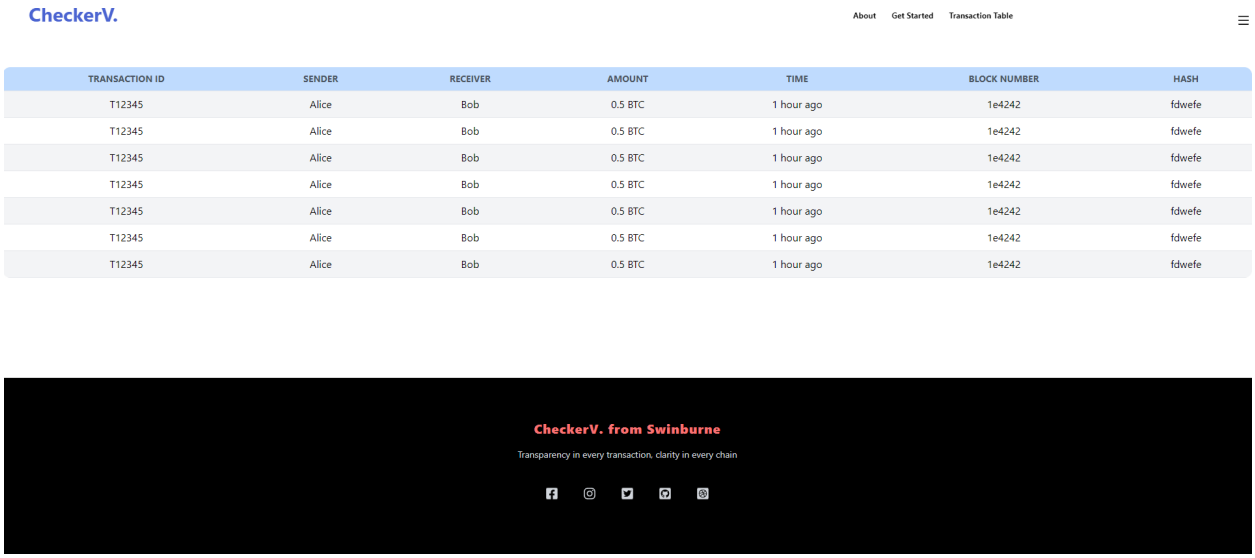


Figure 11: Transaction Page

In addition to the feature described in the device sketches and design drawing, our final website is also responsive, ensuring that users have a consistent experience regardless of the device they use to access the site - desktop, tablet, or smartphone. This adaptability enables users to interact with the platform in a variety of contexts, from in-depth research at a workstation to quick checks on a mobile device (see Figure 12 and Figure 13)

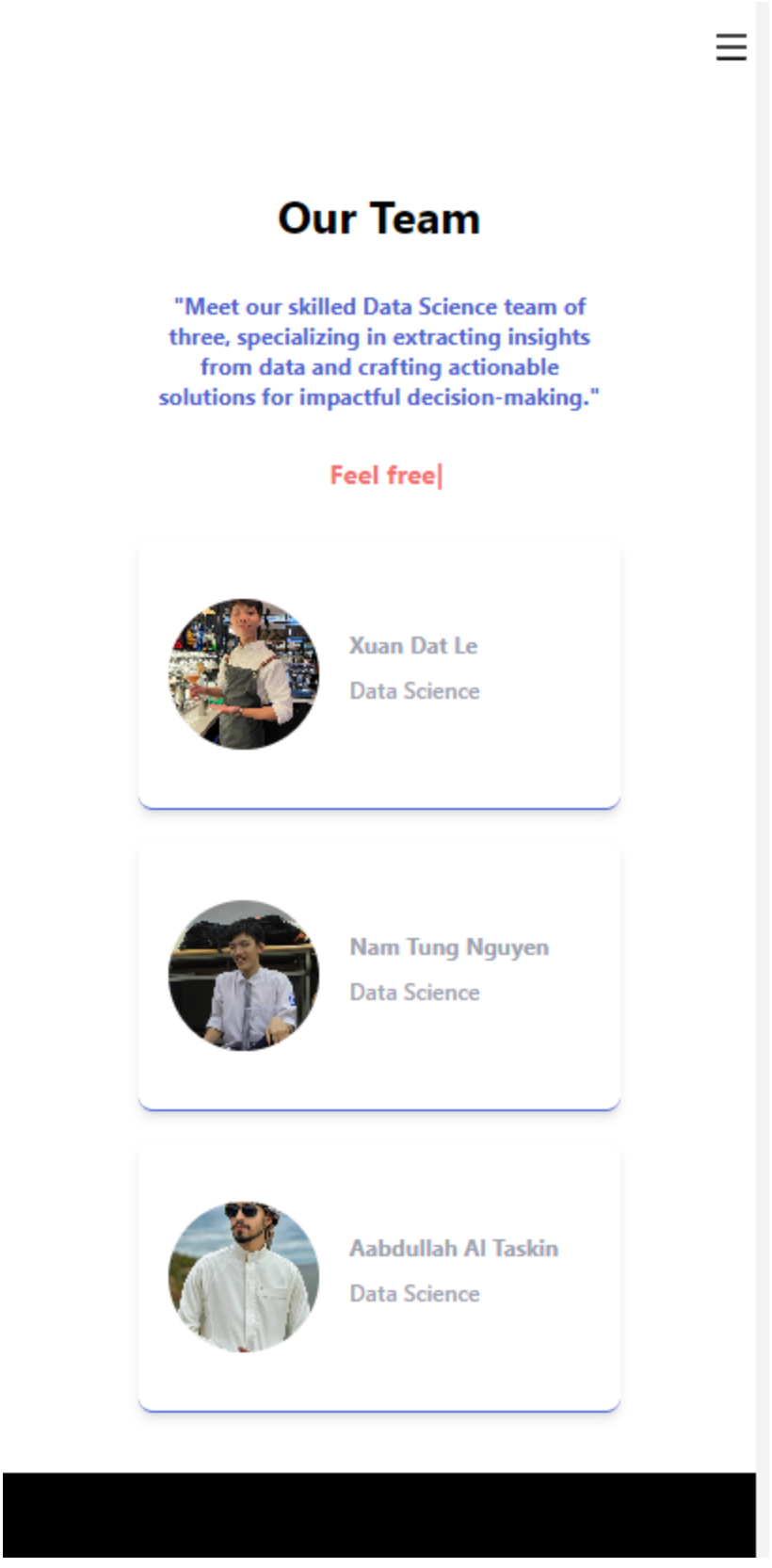


Figure 12: About page in mobile device view

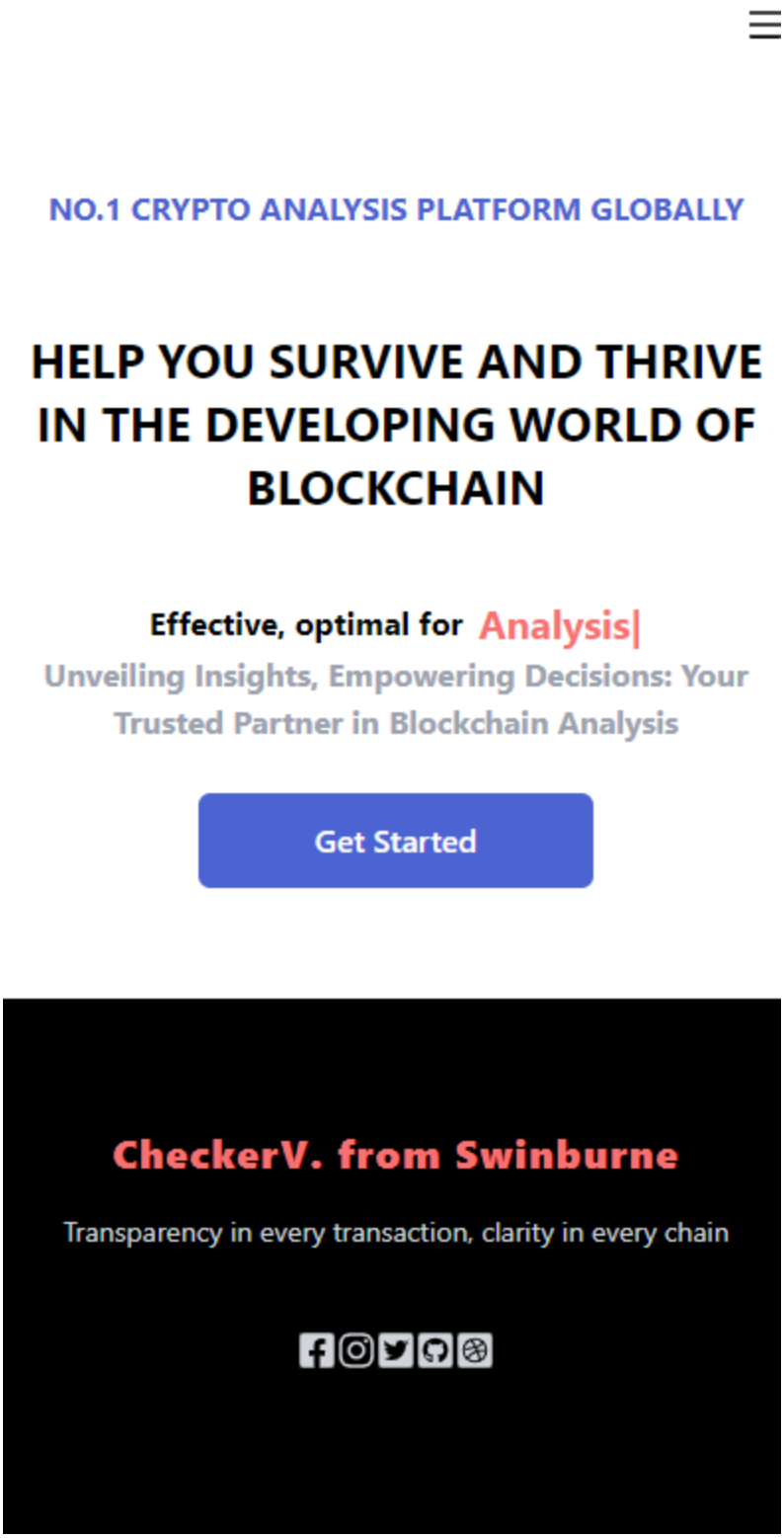


Figure 13 : Home page in mobile device view

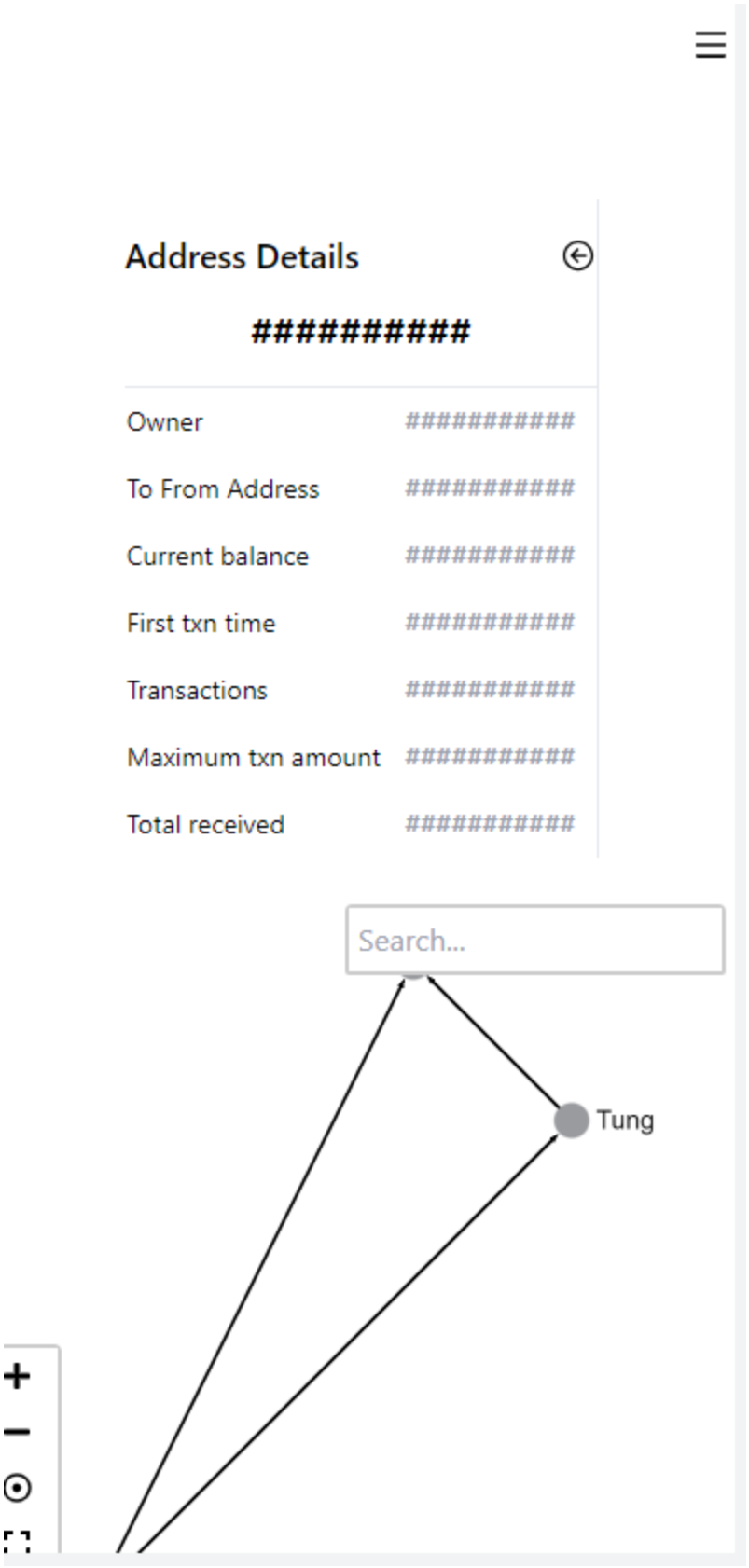


Figure 14: Main page in mobile device view

Last but not least, interactivity has been successfully implemented on the website through the use of a mouse-over effect (see Figure 15) and an interactive feature when the user clicks on a node: the node and its links to other nodes are highlighted, and the information on the address detail is displayed according to the wallet node (see Figure 16). However in this phase of the project, only the owner of the wallet is appropriately displayed.

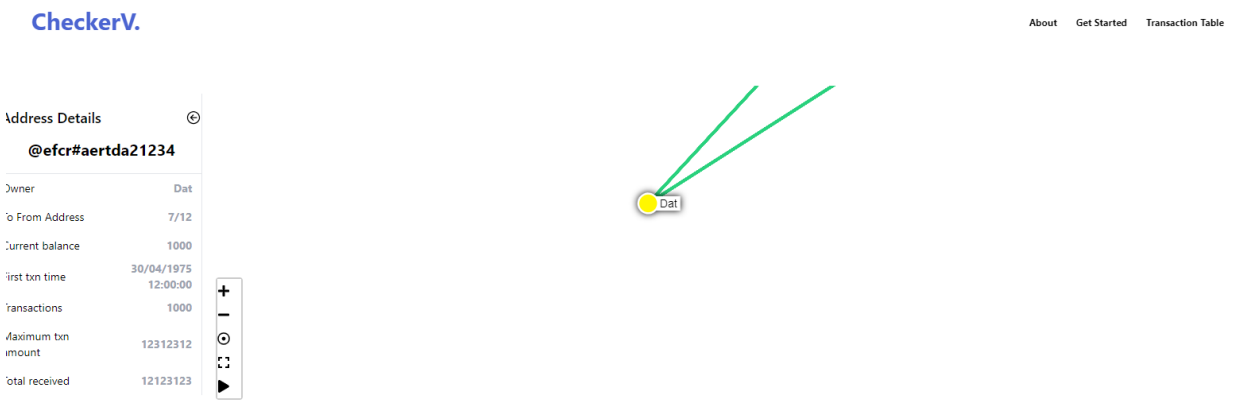


Figure 15 : Mouse over interaction

Figure 16: Interactivity feature of the website: Clicking on a node

4.2. System architecture design

Our system's architecture places paramount importance on performance and scalability. For the website's frontend, we employ ReactJS, a versatile JavaScript library, to create reusable components such as the navigation bar and footer. This approach enhances code organization and maintainability. To further streamline the development process, we utilize ViteJS (Vite, n.d.), known for its rapid hot-reloading capabilities, offering a more agile web development experience.

In addition, our frontend design relies on the capabilities of TailwindCSS (Wathan, n.d.). TailwindCSS offers utility-first classes for design flexibility with a suite of elegantly styled UI components, ensuring our website remains responsive.

For graph visualization, we've chosen SigmaJS (React Sigma, n.d.) - a premier JavaScript library specialized in graph drawing. This makes the presentation of intricate networked data seamless in web applications.

Transitioning to the database layer, Neo4j, a renowned graph database, underpins our platform. It's perfectly suited for representing and querying the interconnected nature of blockchain data, as its node and relationship architecture enables swift data retrieval and adeptly manages intricate queries.

Here is a diagram of the system architecture design of our website

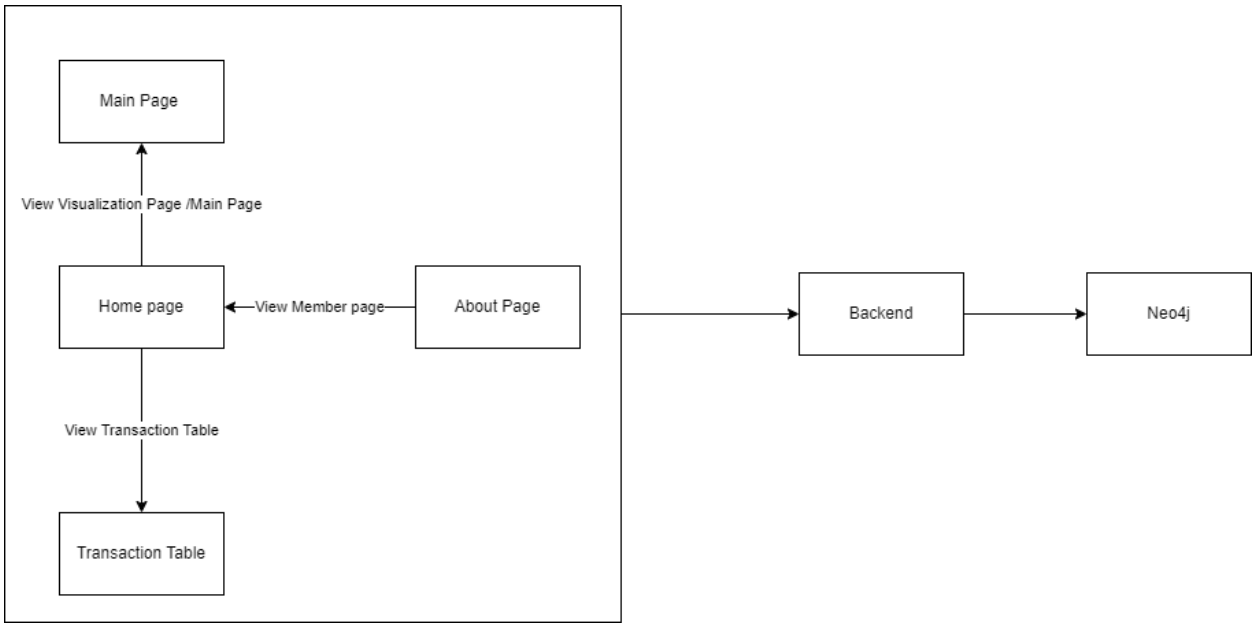


Figure 17: System architecture design diagram

4.3. Backend Database Design

With regard to the graph's database, our group utilized Neo4J graph databases for data storage and retrieval. Neo4j offers an efficient infrastructure for the administration and exploration of interconnected data. In contrast to conventional relational databases which retain data in the form of structured tables, graph databases such as Neo4j are specifically engineered to store, administer, and retrieve information involving data nodes and the relationships between them. The wallet address and the transaction will be represented as nodes in this implementation, where the edges denote the direction.

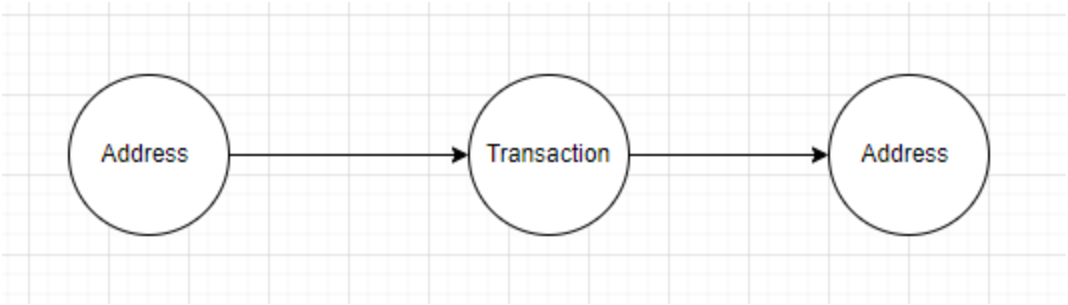


Figure 18: Graph Database Design

The address node uniquely uses the address as its unique key, while the transaction node utilizes the transaction hash as its primary key.

Here are the attributes of the nodes and their data type:

Address Node:

address: string

type: string

Transaction Node:

from_address: string

to_address: string

hash: string

value: string

input: string

transaction_index: integer

gas: integer

gas_used: integer

gas_price: integer

transaction_fee: string

block_number: integer

block_hash: string

block_timestamp: integer

4.4. API Design

API is a fundamental component that enables communication between two applications without requiring any action from the user (www.guru99.com, n.d.). We have developed four distinct APIs for this assignment in order to establish a connection between the front-end and back-end components

4.4.1. Fetch Balance API (Web3 API)

Description

With Web3 API, users can make requests to Ethereum's actual blockchain over the internet, allowing them to interact and retrieve balance information.

Requesting Interaction

This code illustrates the connection to the Ethereum through Infura (see Figure 19) :

```
const web3 = new Web3("https://mainnet.infura.io/v3/6725306487624c2e8da91c6f255f7865");
```

Figure 19: Requesting Interaction for Fetch Balance API

Infura provides a scalable and reliable access point to Ethereum, and the URL is how it is being accessed

Request Parameters

```
const fetchBalance = async (address: string) => {

    const balance = await web3.eth.getBalance(address);
    console.log(typeof(balance))

    return balance;
}
```

Figure 20: Request parameters for Fetch Balance API

address: A string representing the Ethereum address you're fetching the balance for.

Response Format

When calling web3.eth.getBalance(address), it will be resolved with a string representing the balance of the address

For example: 1000000000

4.4.2. Information Fetch API (Neo4J API)

Description

This API allows the retrieval of information of a node or edge when we click on it, utilizing Neo4J Javascript Driver (see Figure 23)

Requesting Interaction

The method session.run() is used to execute Cypher queries and return results.

Query for address info:

```
let result = await session.run('MATCH (n:account {addressId :'+"'"+address+"'"+'}) RETURN n');
```

Figure 21: Query for address info

Query for transfer info by hash:

```
result = await session.run('MATCH (t: transfer {hash :'+"'"+address+"'"+'}) RETURN t');
```

Figure 22: Query for transfer info by hash

Request Parameter

```
const infoFetch = async (address: String) => {
    const driver = neo4j.driver('neo4j+s://c2eda242.databases.neo4j.io:7687', neo4j.auth.basic('neo4j', 'dat12345678'));
    const session = driver.session();
```

Figure 23: Request parameter for Information Fetch API

address: A string representing the Ethereum address or transaction hash you're fetching the info for.

Response Format

When calling session.run(), it will be resolved with a result object which contains data regarding the executed query and an array of records.

Records will contain the actual nodes returned by the query. These nodes can then be parsed to retrieve specific properties.

For example:

```
{
  "records": [
    {
      "_fields": [
        {
          "properties": {
            "addressId": "value1",
            "type": "value2",
            // other properties...
```

```
      }
     }
    ]
   },
 ]
}
```

4.4.3. Fetch Data API (Neo4J Graph Query API)

Description

This API retrieves the node information in the graph database

Requesting Interaction

Utilizing the session.run() method, various types of Cypher queries can be executed on the database:

```
nodes: await session.run('MATCH (n:account {addressId :'+"'"+ address +"'"+'})-[r1]->(t:transfer {from_address: ' + "'" + address + "'" +
rels: await session.run('MATCH (n:account {addressId :'+"'"+ address +"'"+'})-[r1]->(t:transfer {from_address: ' + "'" + address + "'" + '
transfers: await session.run('MATCH (n:account {addressId :'+"'"+ address +"'"+'})-[r1]->(t:transfer {from_address: ' + "'" + address + "'
```

Figure 24: Requesting Interaction for Fetch Data API

Request Parameter

```
const fetchData = async (address: String) => {
    const driver = neo4j.driver('neo4j+s://c2eda242.databases.neo4j.io:7687', neo4j.auth.basic('neo4j', 'dat12345678'));
    const session = driver.session();
```

Figure 25: Request Parameter for Fetch Data API

address: String parameter representing the account address (or another identifier) to fetch related nodes and relationships from the graph database.

Response Format

Responses from the Neo4j driver are in the form of an object containing an array of retrieved records and summary information about the executed query. Records contain arrays of objects or nodes fetched from the database, which can be parsed further to extract and utilize specific data.

For example:

```
{
  "nodes": { "records": [...] },
  "rels": { "records": [...] },
  "transfers": { "records": [...] }
}
```

4.4.4. Transaction Fetch API (Neo4j Transaction Retrieval API)

Description

This API allows developers to interact with a Neo4j database to retrieve blockchain transaction details. Utilizing Cypher query language, it fetches transfer transaction nodes connected to a specified account node in the database.

Requesting Interaction

Query to be executed to retrieve transaction details related to a specified account address:

```
const result = await session.run('MATCH (n:account {addressId :' + "'" + address + "'" + '})-[r1]->(t:transfer {from_address: ' + "'" + addres
```

Figure 26: Requesting Interaction for Transaction Fetch API

Request Parameter

```
const fetchTransactions = async (address: String) => {
    const driver = neo4j.driver('neo4j+s://c2eda242.databases.neo4j.io:7687', neo4j.auth.basic('neo4j', 'dat12345678'));
    const session = driver.session();
```

Figure 27: Request Parameter for Transaction Fetch API

address: String parameter representing the blockchain account address. It's used to identify related transaction nodes in the graph database.

Response Format

Responses from the function would be an object returned by Neo4j containing information about the queried transaction nodes.

For example:

```
{
 "records": [
  {
   "_fields": [
    {
     "properties": {
      "from_address": "0x...",
      "to_address": "0x...",
      // other transaction properties...
     }
    }
   ]
```

```
    },
  ]
}
```

4.5. Functional Description

4.5.1. Search Bar

Description

This search feature enables users to find wallet addresses and transaction hashes.

Use Case

After the user enters the address, a list of relevant addresses will appear for selection. (see Figure 28, Figure 29 and Figure 30)

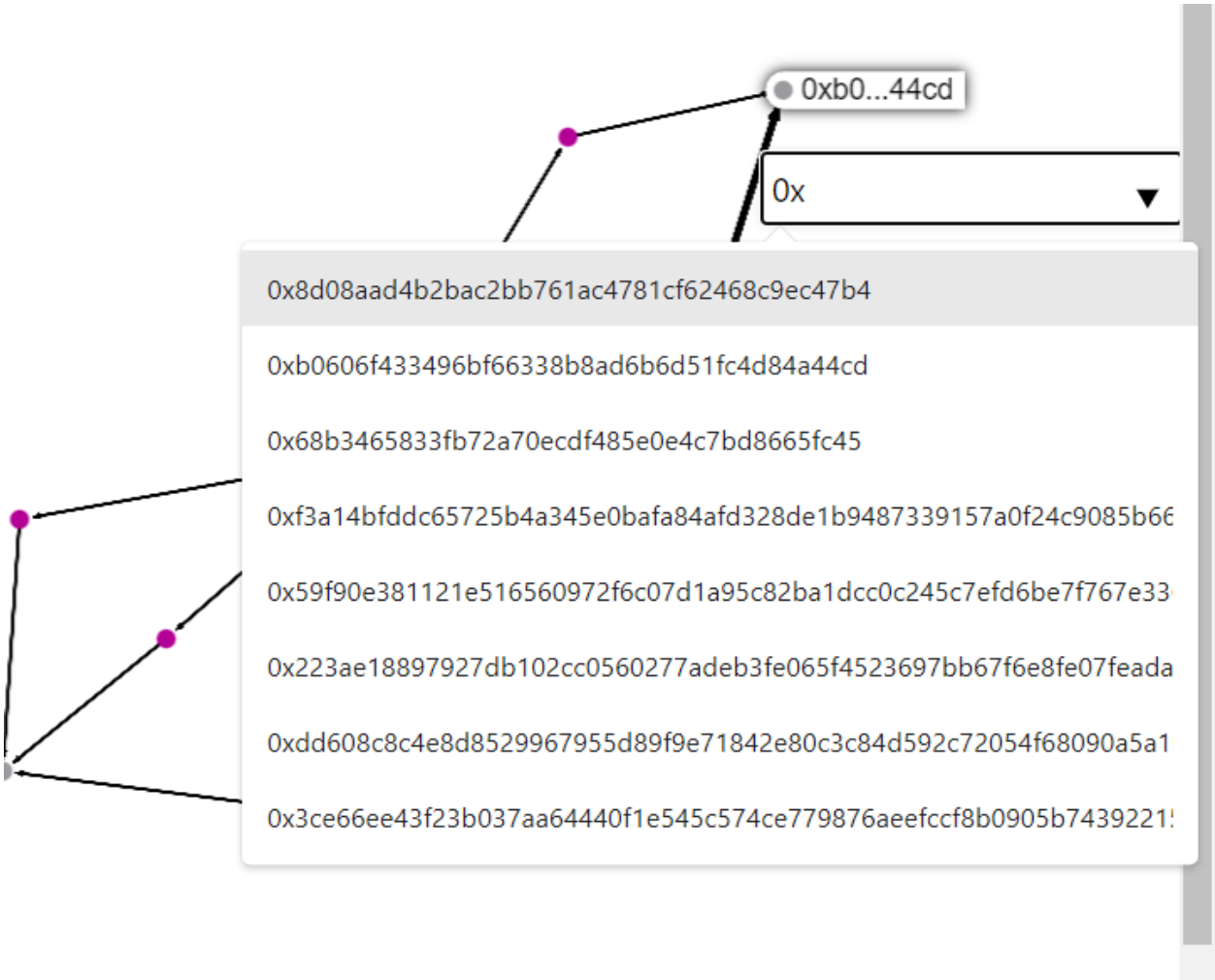


Figure 28: Searching functionality

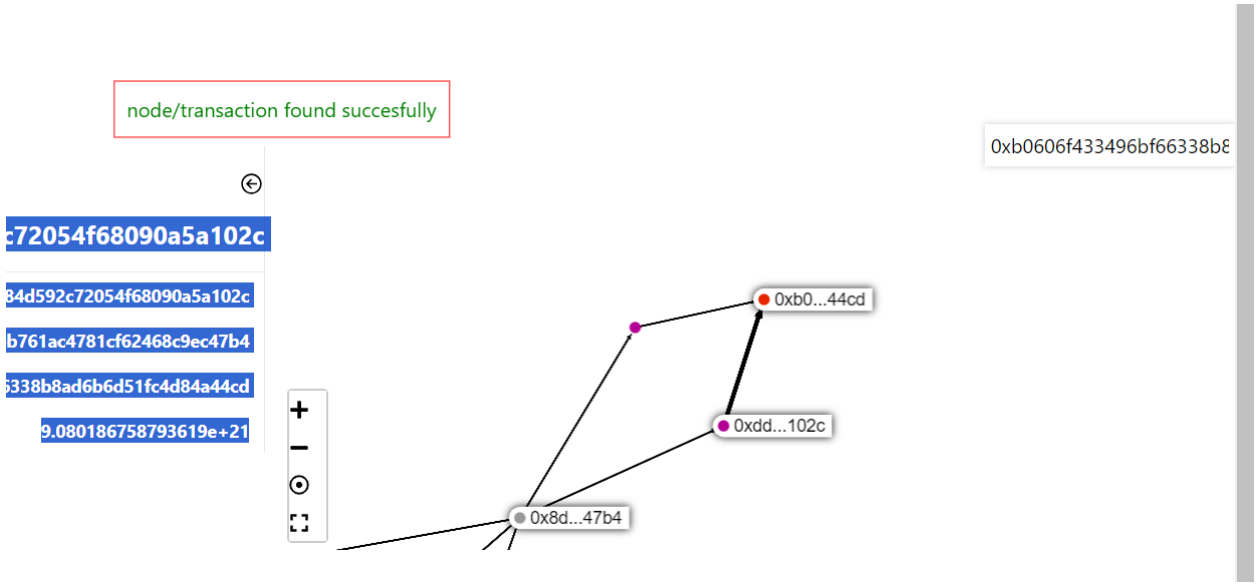


Figure 29: Node/transaction found successfully (with message)

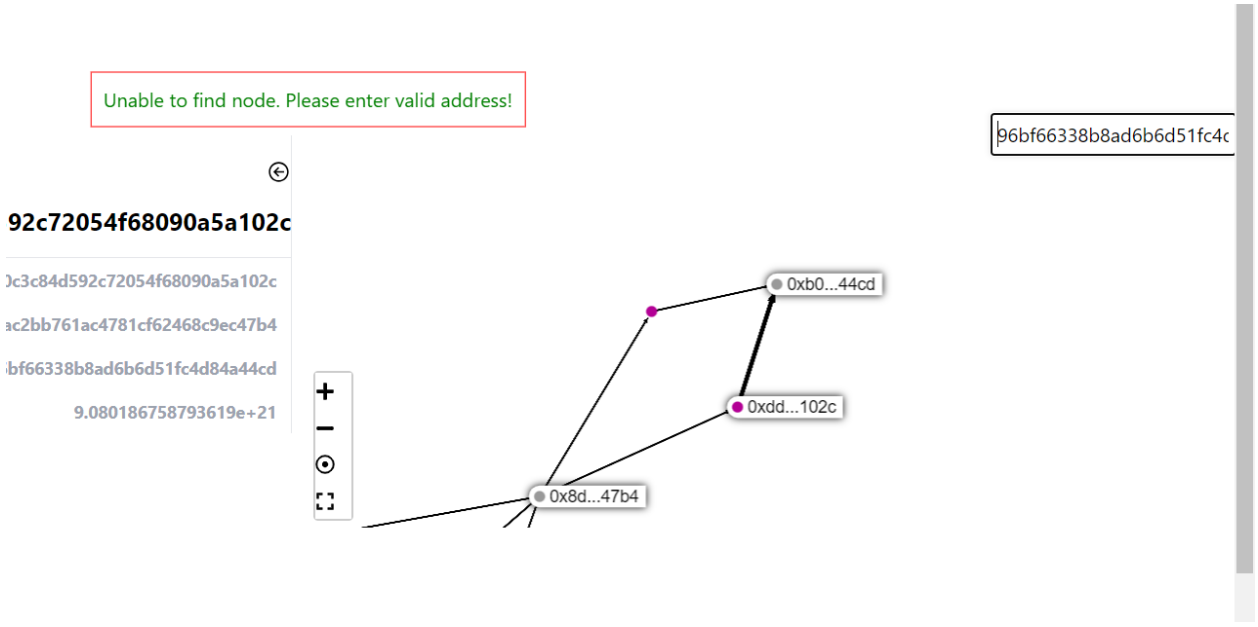


Figure 30: Unable to find node

4.5.2. Information Table

Description

The information table presents data pertaining to the edges or nodes.

Use Case

When a user clicks on a node or edge, the associated information will be displayed. (see Figure 31, Figure 32 and Figure 33)

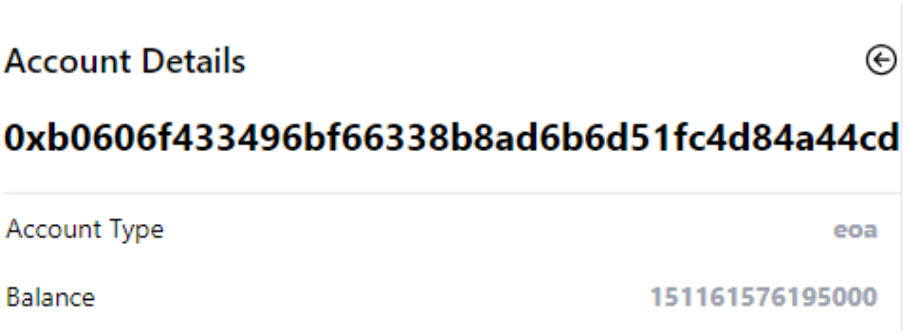


Figure 31: Information Table

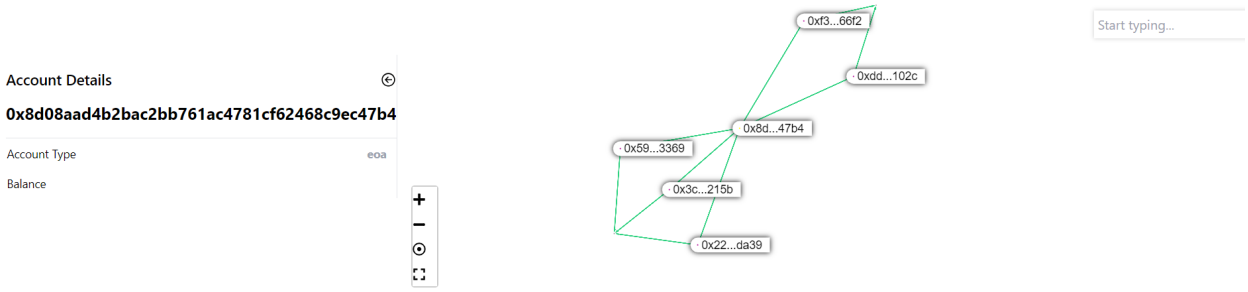


Figure 32: Information Table when users click on a node

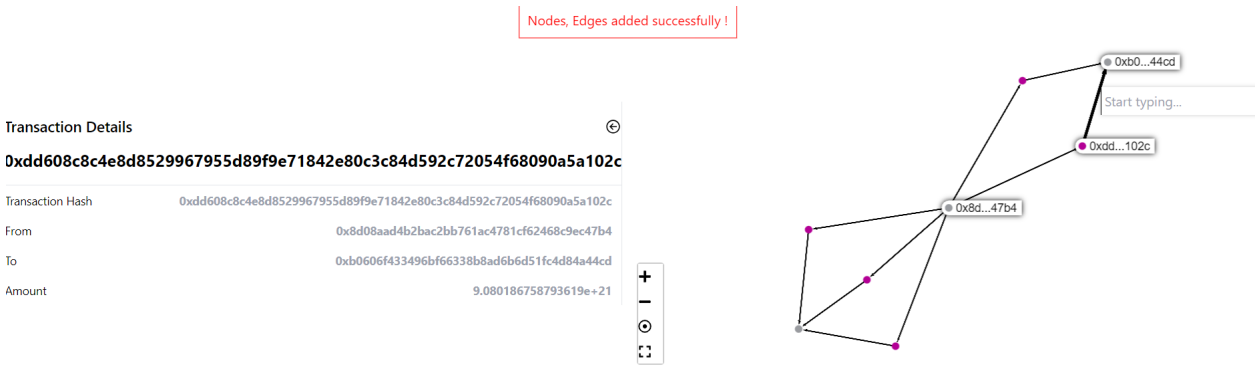


Figure 33: Information Table when users click on a transaction

4.5.3. Transaction Graph

Description

This graph illustrates the wallet address's transaction patterns.

Use Cases

Users have the ability to explore the connected addresses by clicking on the address nodes. Address nodes are coloured gray, whereas transaction nodes are coloured purple. (see Figure 34 and Figure 35)

Figure 34: Initial graph visualization

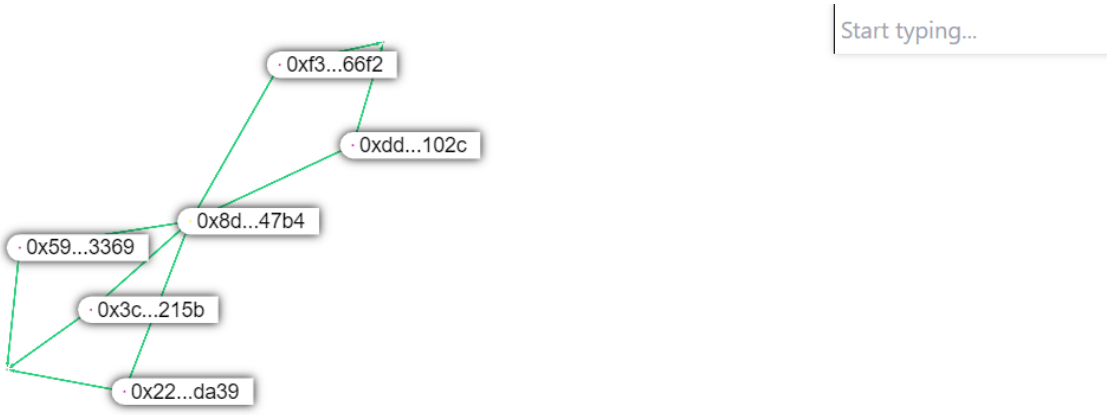


Figure 35: Visualization expanded after clicking on an address node

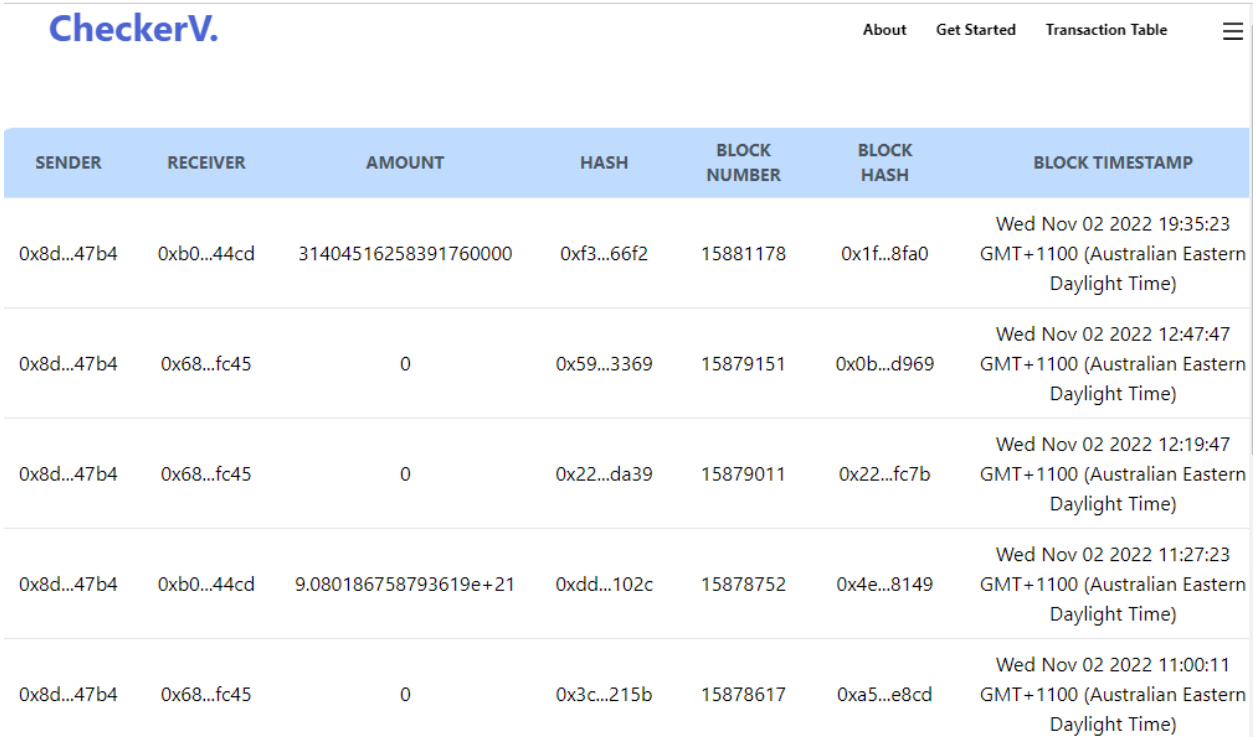
4.5.4. Transaction Table

Description

This table display the information related to the transactions

Use Case

When the users click on a node or edge, the transaction table will be expanded (see Figure 36 and Figure 37)



SENDER	RECEIVER	AMOUNT	HASH	BLOCK NUMBER	BLOCK HASH	BLOCK TIMESTAMP
0x8d...47b4	0xb0...44cd	31404516258391760000	0xf3...66f2	15881178	0x1f...8fa0	Wed Nov 02 2022 19:35:23 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x59...3369	15879151	0x0b...d969	Wed Nov 02 2022 12:47:47 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x22...da39	15879011	0x22...fc7b	Wed Nov 02 2022 12:19:47 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0xb0...44cd	9.080186758793619e+21	0xdd...102c	15878752	0x4e...8149	Wed Nov 02 2022 11:27:23 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x3c...215b	15878617	0xa5...e8cd	Wed Nov 02 2022 11:00:11 GMT+1100 (Australian Eastern Daylight Time)

Figure 36: Initial transaction table

0x8d...47b4	0xb0...44cd	31404516258391760000	0xf3...66f2	15881178	0x1f...8fa0	Wed Nov 02 2022 19:35:23 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x59...3369	15879151	0x0b...d969	Wed Nov 02 2022 12:47:47 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x22...da39	15879011	0x22...fc7b	Wed Nov 02 2022 12:19:47 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0xb0...44cd	9.080186758793619e+21	0xdd...102c	15878752	0x4e...8149	Wed Nov 02 2022 11:27:23 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0x68...fc45	0	0x3c...215b	15878617	0xa5...e8cd	Wed Nov 02 2022 11:00:11 GMT+1100 (Australian Eastern Daylight Time)
0x8d...47b4	0xb0...44cd	31404516258391760000	0xf3...66f2	15881178	0x1f...8fa0	Wed Nov 02 2022 19:35:23 GMT+1100 (Australian Eastern Daylight Time)
						Wed Nov 02 2022 12:47:47

Figure 37: Expanded transaction table

4.6. Project Deployment Instruction

It is very straightforward to deploy this project. Users only need to follow this two steps

Step 1: Users must download all the required libraries using the command

npm install

Step 2: Users can see the website using the command

npm run dev

Note: In case that the users are unable to connect to the database, there are several steps that they need to do:

Step 1: Open Neo4J Aura

Step 2: Log in to Neo4J Aura using the following account

Email: 103487949@student.swin.edu.au

Password: Datdz090104

Step 3: Click on Open to connect to instance

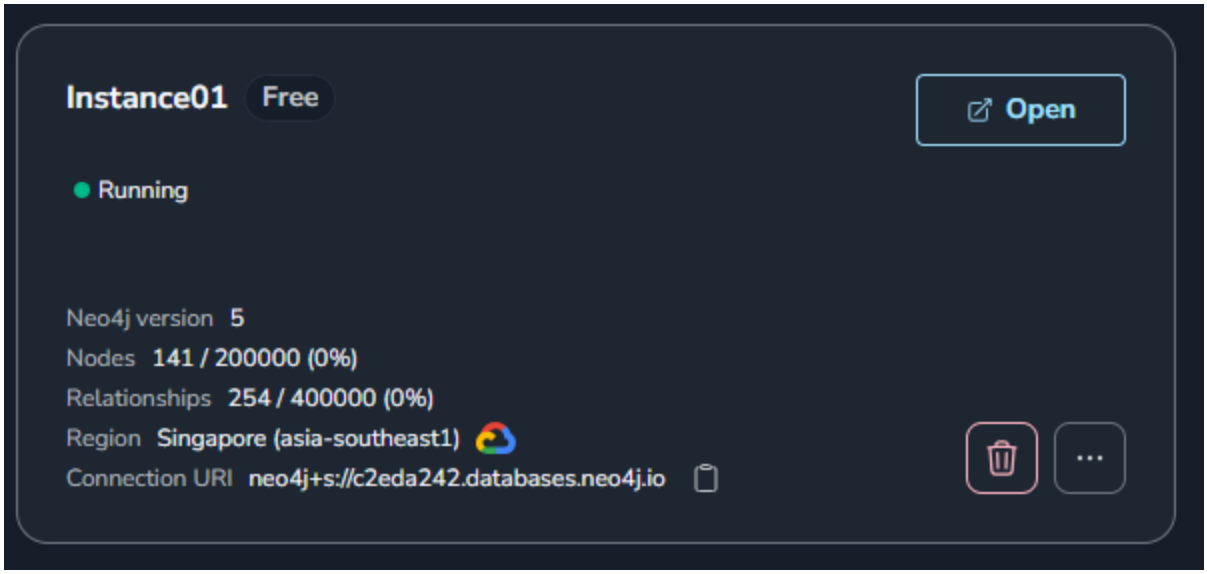


Figure 38: Database instance

Step 4: Enter the information to connect to Database

Database user: neo4j

Password: dat12345678

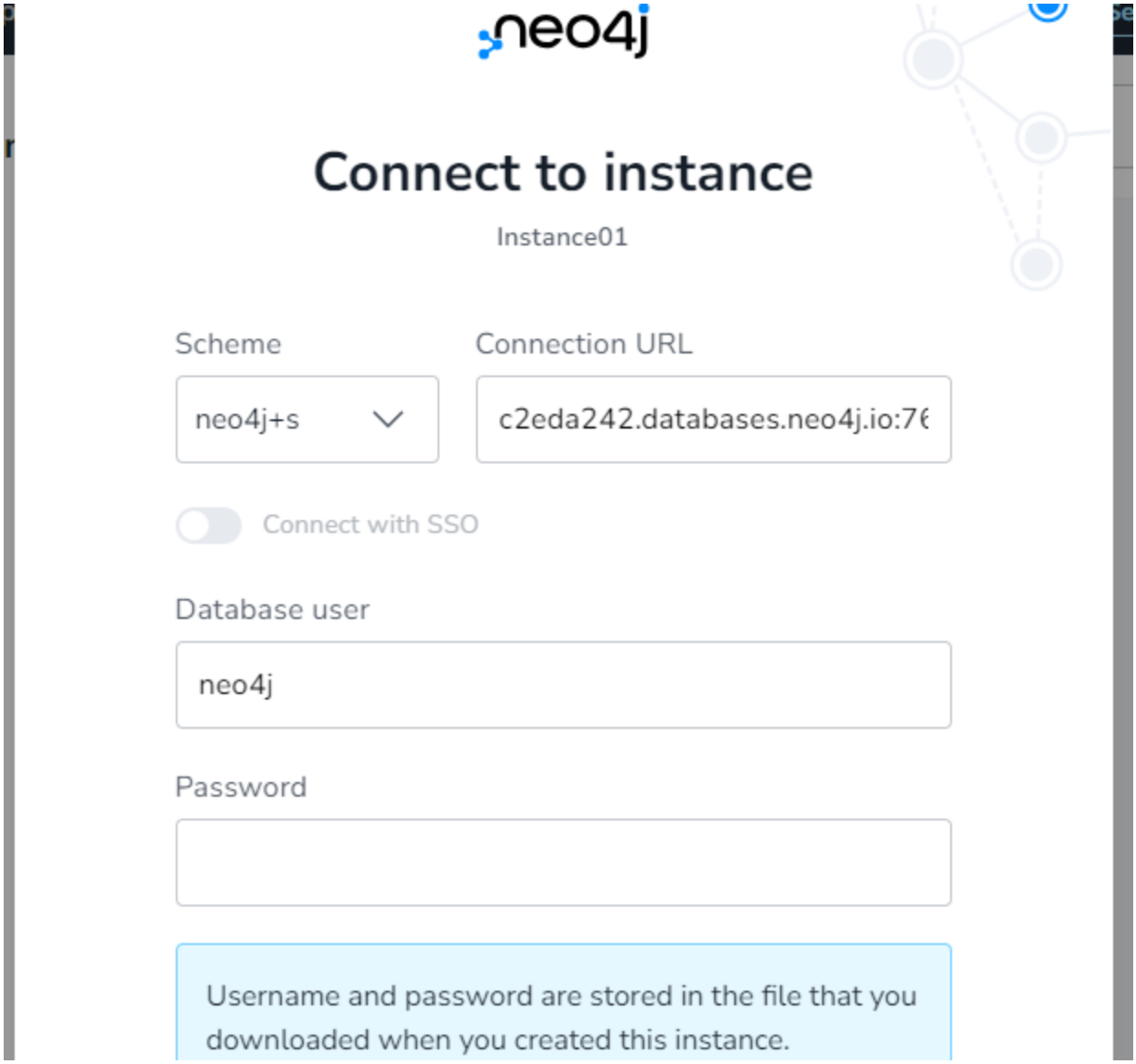


Figure 39: Connect to instance

5. Conclusion

In conclusion, our team has developed a user-friendly Blockchain Transaction Visualisation System Website, which demonstrates our commitment to transform complex blockchain data into a practical visual representation. Despite the fact that we have faced numerous obstacles due to our lack of expertise in website development, we have diligently worked to enhance our project by ensuring stability and reliability through the efficient use of graph databases, and by implementing various functionalities and features.

6. Reference

1. Hayes, A. (2023). Blockchain Facts: What Is It, How It Works, and How It Can Be Used. [online] Investopedia. Available at: https://www.investopedia.com/terms/b/blockchain.asp.
2. React Sigma. (n.d.). Available at: https://sim51.github.io/react-sigma/.
3. Vite. (n.d.). Available at: https://vitejs.dev/.
4. Wathan, A. (n.d.). [online] tailwindcss. Available at: https://tailwindcss.com/.
5. Linkurious. (n.d.). Graph visualization: why it matters. [online] Available at: https://linkurious.com/blog/why-graph-visualization-matters/.
6. Soegaard, M. (n.d.). The Power of White Space in Design. [online] The Interaction Design Foundation. Available at: https://www.interaction-design.org/literature/article/the-power-of-white-space#:~:text=White%20space%20can%20help%20guide.
7. redballoon.in. (n.d.). what does navigation mean in a website | user friendly navigation. [online] Available at: https://redballoon.in/what-does-navigation-mean/#:~:text=What%20is%20User%20Friendly%20navigation.
8. www.guru99.com. (n.d.). What is an API? Meaning, Definition, Types, Application, Example. [online] Available at: https://www.guru99.com/what-is-api.html.