1. **Chapter Title: Profile Clone Detection and Impact Analysis on Social Media Platforms like LinkedIn**

**Table of Contents**

1. **Introduction**
2. **Background and Literature Review**
3. **Methodology**
4. **Tools and Technologies Used**
5. **Dataset**
6. **Environmental Set Up**
7. **System Architecture**
8. **Implementation Details**
9. **Results and Discussion**
10. **Future Work**
11. **Novel Aspects**

**Introduction:**

Linkedin is one of the most famous social media platforms, especially in professional networking for individuals and organizations. It provides a platform for job seekers and employers to connect and get mutual benefit . As such the security and integrity of this platform is of significant importance.

During recent years however, there has been an increase in a certain subset of fake profiles called “Clone Profiles”. These profiles aim to impersonate an existing user by having similar names, profile pictures, and various such attributes. Propagation of these clone profiles is dangerous as it can harm the legitimate user and cause professional harm to him/her.

As such it is important to detect these clones, and analyze their behavior through the network to get some idea on the damage done by them. By this we mean, getting a general idea on which profiles the clone profiles have interacted with, or studying the reach of the clone profile in the network. Our project aims at simulating a network of nodes that is similar to LinkedIn where addition of new nodes and connections is possible.

A clone detection target profile is obtained. Textual fields of the profile attributes are compared and analyzed using deep learning string matching algorithms. Impact analysis of detected potential clones is done using graph network analysis.

**Background and Literature Review:**

**PAPER 1**: Detecting Social Network Profile Cloning

[2] Author: Kontaxis et al.

In this paper the authors have developed a tool for finding clone profiles. The tool has three components, the Information Distiller, Profile Hunter and Profile Verifier. The first component was used to gather user data from the network. The Profile Hunter was used to find profiles that could belong to the user and Profile Verifier was used to find similarity between the given profiles.Exact string matching was used for the text comparison and ImageMagick suite for naive image comparison. The tool was able to detect profile clones without any false negatives or false positives. Manual inspection verified that the score detected by Profile verifier was accurate.

Takeaway: This paper primarily talks about a tool that was used to find clone profiles. This served as a foundational base to our current architecture and helped us outline the necessary components for the project. The tool used in the paper uses exact string matching between the profiles, we aimed to enhance this by using sentence transformers for semantic string comparison.

**PAPER 2**: Exploring machine learning techniques for fake profile detection in online social networks [3]

Author: Bharti et al.

In this paper the authors have performed a comparative analysis on the various Machine Learning models that have been used for the purpose of fake profile detection. It is a very detailed study, and gives insight into the advantages and disadvantages of each model.Some of the techniques surveyed are: Random Forest, SVM5 , GAN 9 s and Deep Learning Models. The results showed that Supervised machine learning models were used by most researchers for fake profile detection. Additionally some of the models did not take into account the temporal nature of the data or were only trained on a very small subset of data.

Takeaway: This paper compares various machine learning models that were used to detect fake profiles. It helped us understand the various advantages and disadvantages of each of the existing machine learning models

**PAPER 3**: Performance Comparison Analysis of ArangoDB, MySQL, and Neo4j: An Experimental Study of Querying Connected Data [1]

Author: Johan Sandell, Einar Asplund, et al.

Testing workloads against database-specific benchmark tools allowed for the collection of performance data tailored to each technology's characteristics. Baseline performance measurement, crucial for comparison, was established as the machine's performance in the absence of any queries, providing a reference point for analysis The results suggest that graph databases (Neo4j and ArangoDB) outperform the relational database (MySQL) in specific aspects when dealing with big data with connected relationships.

Takeaway: This paper compares the performance of Neo4j ,ArangoDB and MySQL and the results suggest that Neo4j outperforms MySQL when it comes to dealing with data that has connected relationships. Our project primarily deals with social network profiles that are all connected to each other, and this can be visualized as a network graph. Hence, we chose Neo4j to create our network and this paper aided that thought process.

**METHODOLOGY:**

**Steps implemented:**

1. Preprocess Data: This entity takes in a dataset as input and returns a preprocessed dataset as output. This step processes linkedin data which includes steps like cleaning the data , handling missing values and transforms into a form that is suitable for clone detection analysis.
2. Add Data into Network Graph(Neo4j) as Node: This entity takes in preprocessed data as input and adds the data as nodes into the Graph DB and hence the output is an updated Graph DB.
3. Profile Comparison: This entity takes in User profile and Other profile data as input and performs a profile to profile comparison using Sentence Transformers and Levenshtein algorithm and produces a set of potential clones as output.
4. Flagging Clones: This takes in the User profile data and Clones Flagged by user as input and performs an analysis based on Centrality measures, Time of creation of nodes and Common neighbors among the User profile and flagged clones and returns a decision or result as output.
5. Visualization Layer(UI): The Graph, flagged clones list and the impact assessment results are displayed on the front end.

### **Tools Used and Reasons for Use:**

* **Neo4j: Graph Database for Simulating the Social Network**
  + **Reason for Use**:
    - Neo4j is a graph database that is well-suited for representing **social network structures**, where entities (e.g., user profiles) are connected to one another. Social networks are naturally modeled as graphs, with nodes representing profiles and edges representing relationships such as connections or interactions.
    - It enables **fast traversal of relationships** between profiles supporting various types of **graph queries** to efficiently compute connections, neighbors, and centrality scores, which are essential in analyzing the impact of a cloned profile on the network.
    - Additionally, Neo4j offers **native support for graph algorithms** like centrality measures (degree, betweenness, eigenvector, etc.) that are pivotal for determining the influence and reach of a profile in the network. Its **visualization tools** make it easier to represent and analyze connections in a large dataset.
    - Additionally, Neo4j's **scalability** supports up to **200,000 nodes** and **400,000 relationships**, making it particularly suitable for creating a large-scale network like LinkedIn.
* **Python: Backend Implementation**
  + **Reason for Use**:
    - Python is one of the most widely-used programming languages for **data science** and **machine learning**, making it the ideal choice for implementing algorithms like **profile comparison**, **centrality metrics calculation**, and **string matching**.
    - It provides an extensive ecosystem of libraries such as **NetworkX** for graph analysis and **Hugging Face** for natural language processing. Python’s simplicity, readability, and rich set of libraries make it efficient for handling backend operations, querying Neo4j, and processing profile data.
    - Python also supports **multithreading** and **asynchronous programming**, which helps to manage multiple network requests and large data handling efficiently.
* **Flask: Backend Framework for Server-Side Functionality**
  + **Reason for Use**:
    - Flask is a lightweight and flexible **web framework** that is perfect for building **RESTful APIs** for server-side functionalities. It acts as the bridge between the backend (where data is processed and analyzed) and the frontend (where users interact with the application).
    - Its simplicity and minimalistic structure allow for easy integration with other tools like **Neo4j** and **ReactJS**, making it ideal for this project’s requirements. Flask is also highly customizable, allowing us to manage data flow between the **graph database**, **machine learning models**, and **user interface** efficiently.
* **ReactJS: Frontend Framework for Building the User Interface**
  + **Reason for Use**:
    - ReactJS is a modern **JavaScript library** for building dynamic and interactive **user interfaces**. It enables the creation of responsive and fast user interfaces for displaying **profile comparison results**, **graph visualizations**, and **impact assessments**.
    - It is component-based, which allows for **reusable UI components** and a streamlined development process. React also efficiently handles state management, making it easier to update the user interface based on backend results like flagged clone profiles and centrality scores.
    - Since visualizing data like **graph networks** and impact analysis requires a fast and efficient frontend, ReactJS’s ability to handle dynamic updates and **real-time rendering** is essential.
* **Kaggle Datasets: Used for Initial Data Collection and Experimentation**
  + **Reason for Use**:
    - Kaggle provides a rich collection of **public datasets** for experimentation and analysis, which is valuable for testing the clone detection system. The **LinkedIn User Profiles Dataset** from Kaggle was used as a proxy for real-world data, simulating a professional social network environment.
    - This dataset contains structured information, such as names, job positions, education, and descriptions, which are the key features needed to compare profiles and identify clones. The availability of diverse attributes makes the dataset an ideal choice for testing **string matching** and **semantic similarity** models.
    - Kaggle datasets also come in formats that are easy to process (CSV, JSON, etc.), making them suitable for data preprocessing and experimentation.
* **Levenshtein Algorithm: Exact String Matching for Profile Attribute Comparisons**
  + **Reason for Use**:
    - The **Levenshtein algorithm** computes the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. This algorithm is perfect for **exact string matching**, helping to detect profiles with **minor variations** in attributes like name, username, or job title.
    - It is computationally efficient for small text fields, making it suitable for profile comparisons where slight variations, such as spelling errors or abbreviations, could indicate cloned profiles.
    - Using the Levenshtein distance allows the system to quickly identify potential clones with **exact or near-exact matches**, contributing to the final similarity score in combination with semantic analysis.
* **Sentence Transformers: Hugging Face Model for Semantic String Matching**

**Reason for Use**:

* + To help find semantic similarity between the profiles and as such detect potential clones in a better manner.
  + Sentence transformers have been trained on a large corpus of data and have also been trained on multiple languages thus making it an efficient model to use.
  + Sentence transformers are efficient for large datasets as embeddings for each profile can be generated independently. This makes it useful when it comes to scalability.
  + It is able to handle large texts more efficiently as compared to models like BERT thus making it useful for profile comparison. A given profile can have a large number of fields.
  + It uses independent encoding , which reduces computational cost. This is especially useful when we have a large dataset, this basically reduces the number of comparisons
* **vis-network: A JavaScript Library for Visualizing and Interacting with Network Graphs**
  + **Reason for Use**:
    - **vis-network** is an interactive library designed for **visualizing graphs and networks** in web applications. Given that the system deals with a **social network of profiles**, visualization is key to understanding the relationships between nodes (profiles) and edges (connections).
    - This library allows the **dynamic exploration of connections**, making it easy for users to visually assess the impact of clone profiles and navigate through their neighbors, shared connections, and influence in the network.
    - It provides an intuitive interface for users to **interact with the graph**, such as zooming in on nodes, clicking on nodes to view profile details, and displaying flagged clones, making the overall user experience more engaging and informative.

**Dataset:**

The LinkedIn Professional Profiles Dataset is a dataset containing attributes of LinkedIn profiles in the United States Some of the attributes involved in the dataset are:

The mandatory fields in Linkedin are **First Name**, **Last Name**, and **Country**, with **Industry** being a private field.

Out of the total 31 features available in the LinkedIn dataset, we are utilizing 6 key features: timestamp, name, username, position, about, and education. These features were chosen for their relevance in comparing profiles and detecting potential clones.

Key attributes into consideration:

Name: Full name of the professional.

Username: LinkedIn username of the professional.

Position: Detailed job position within the company.

About: Description summarizing the professional’s expertise and goals

Education: Educational background and institutions attended.

Dataset (created in 2023):

<https://www.kaggle.com/datasets/manishkumar7432698/linkedinuserprofiles>

### **Environmental Setup**

To ensure that the application functions correctly, it's important to use the basic minimum versions of the software and tools.

**1. Development Environment**

* **Operating System**:
  + **Primary OS**: Ubuntu 20.04 LTS (preferred for stability and compatibility)
  + **Alternate OS**: Windows 10 (or later) with Windows Subsystem for Linux (WSL) for Linux-based tools
* **IDE and Code Editors**:
  + **Visual Studio Code (VS Code)**: Version 1.60.0 or later
  + **PyCharm**: Community Edition 2021.2 or later (Professional Edition is optional for advanced features)
* **Version Control**:
  + **Git**: Version 2.31.0 or later
  + **GitHub**: Use GitHub.com for hosting the repository; GitHub CLI or desktop application can be used for convenience

#### **2. Software and Tools**

* **Neo4j**:
  + **Minimum Version**: Neo4j 4.x
  + **Setup**: Install Neo4j Desktop 4.x or use Neo4j Aura (cloud-based service). Ensure adequate configuration for memory and storage.
* **Python**:
  + **Minimum Version**: Python 3.7
  + **Libraries**:
    - **NetworkX**: Version 2.6.3 or later
    - **Sentence Transformers**: Version 2.2.0 or later
    - **Levenshtein**: Version 0.12.2 or later
* **Flask**:
  + **Minimum Version**: Flask 2.0.1
  + **Setup**: Install Flask and necessary extensions using pip.
* **ReactJS**:
  + **Minimum Version**: React 17.x
  + **Setup**: Use Create React App or a similar boilerplate to set up the project. Node.js 14.x or later, and npm 6.x or later are required.
* **Kaggle Datasets**:
  + **Dataset**: LinkedIn User Profiles dataset from Kaggle; ensure dataset is in a compatible format (CSV or JSON).
* **vis-network**:
  + **Minimum Version**: vis-network 6.6.2
  + **Setup**: Install via npm (npm install vis-network) or include via a CDN in the frontend application.

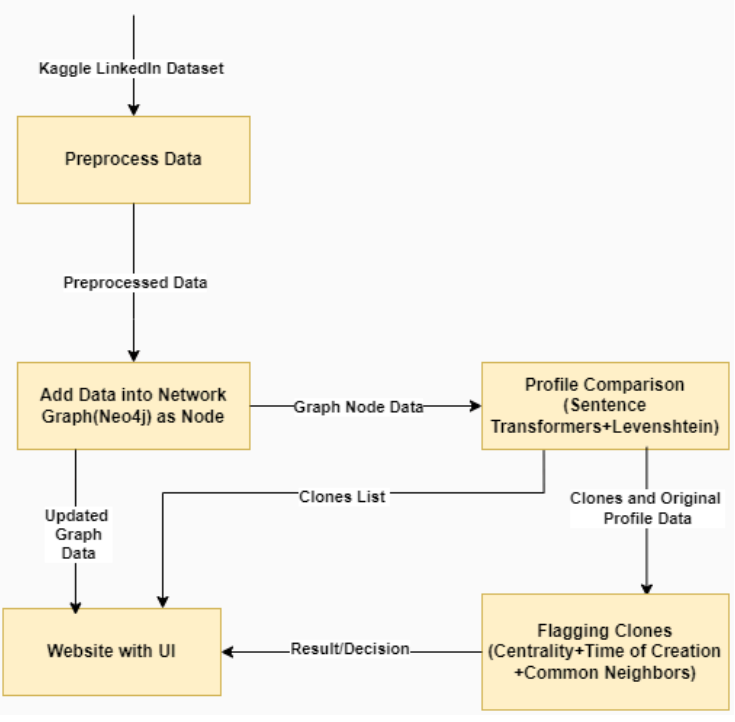
#### **3. Hardware Requirements**

* **Development Machine**:
  + **CPU**: Multi-core processor (e.g., Intel i5/i7 or AMD Ryzen).
  + **RAM**: Minimum 16 GB, preferably 32 GB for handling large datasets and multi-threaded operations.
  + **Storage**: SSD with at least 256 GB of free space for faster read/write operations and handling database storage.
* **Server Environment** (For deployment):
  + **CPU**: Multi-core processor with high clock speed.
  + **RAM**: Minimum 32 GB, with higher capacity based on dataset size and user load.
  + **Storage**: SSD or NVMe storage with sufficient capacity (1 TB or more) for large-scale data storage and backups.

#### **4. Network**

* **Network Configuration**:
  + Ensure secure access to Neo4j and Flask servers, with proper firewall rules and access controls.
  + Use HTTPS for secure communication between the frontend and backend.

**Architecture/ methodology in detail**



**System Architecture**

**1. Data Preprocessing**

Before we can detect clone profiles, the raw LinkedIn dataset needs to be preprocessed to ensure the data is in a clean and structured format. The preprocessing stage involves the following steps:

Data Cleaning: Removing irrelevant information, handling missing values, and ensuring uniformity across fields (such as standardizing name formats).

Feature Selection: Selecting important fields from the LinkedIn dataset, such as Name, Username, Position, About, and Education, to focus on attributes most relevant for clone detection.

Output: A preprocessed dataset with clean and structured LinkedIn profiles that are ready to be added to the network graph for clone detection.

**2. Adding Data to Network Graph (Neo4j)**

In this stage, the cleaned and processed LinkedIn profiles are inserted into a graph database (Neo4j). Each LinkedIn profile is represented as a node in the graph, while connections between users (e.g., colleagues, connections) are modeled as edges between nodes.

1. Node Creation: For each LinkedIn profile, a corresponding node is created in Neo4j with key attributes such as Name, Position, and Education.
2. Edge Creation: Edges (relationships) are added to the graph, representing connections between users. These could include co-worker relationships, previous employers, or shared academic history.

The choice of Neo4j is essential here because social networks are inherently graph-like in nature, where connections and relationships are the focus. Neo4j enables easy querying and visualization of these relationships, making it ideal for this type of analysis.

Output: A Neo4j graph database containing LinkedIn profiles as nodes, connected by relationships (edges) that represent user connections.

**3. Profile Comparison**

The core of the clone detection process involves comparing user profiles to detect potential clone profiles. This is done using a hybrid approach combining:

1. Levenshtein Algorithm: This algorithm performs exact string matching to detect minor variations between profile attributes like name or job title. It works by calculating the minimum number of edits (insertions, deletions, or substitutions) required to turn one string into another.
2. Sentence Transformers: These are used for semantic string matching, meaning profiles that might have similar meanings but are written differently can still be identified. For example, "Software Developer" and "Developer in Software" would be semantically similar. Sentence Transformers provide embeddings for each textual profile field, which allows the model to understand the underlying context and meaning.

Similarity Score Calculation: The results from both Levenshtein and Sentence Transformers are combined to calculate an overall similarity score for each profile pair. If the similarity score exceeds a threshold, the profile is flagged as a potential clone.

Output: A set of potential clone profiles with calculated similarity scores.

**4. Flagging Clones and Impact Analysis**

Once potential clone profiles have been detected, the next step is to analyze their behavior and influence in the network through centrality metrics:

Centrality Measures: Using the NetworkX library, several centrality measures are computed:

1. Degree Centrality: Number of connections a node (profile) has in the network.
2. Closeness Centrality: How close a node is to all other nodes in terms of connection paths.
3. Betweenness Centrality: How often a node appears on the shortest path between other nodes.
4. Eigenvector Centrality: Influence of a node based on its connections to other high-scoring nodes.

These centrality metrics help in understanding the impact and reach of clone profiles in the network. For example, a clone profile with high degree or eigenvector centrality could have a significant influence in spreading false information or misrepresenting a legitimate user.

Common Neighbors: The number of shared connections between the original profile and the clone profile is analyzed to understand network overlap. This can provide insights into how far the clone profile has infiltrated the user’s network.

Output: A report of the flagged clone profiles, including their centrality scores and shared connections.

**5. Visualization Layer (UI)**

To present the analysis and findings, a user interface (UI) is developed, which provides a visual representation of the clone detection and impact analysis results. This involves:

1. Graph Visualization: The graph, representing user profiles and connections, is visualized using libraries like vis-network, allowing users to interact with the network and explore clone profiles and their connections.
2. Flagged Clones List: A list of clone profiles, along with their similarity scores and centrality measures, is displayed.
3. Impact Assessment Results: Graph-based analysis results, such as the influence of the clone profile on the network, are presented for further exploration.

Output: A front-end interface displaying clone profiles, impact analysis results, and the network graph for easy interpretation.

**6. Notifying Neighbors**

After detecting clone profiles, it is crucial to notify the legitimate profile’s connections to prevent further harm. This step is done only after a thorough meticulous verification has been done by the admin to confirm about the fakeness of the flagged clone. This step involves:

Sending Notifications: Establishing a CLONE\_NEIGHBOUR relationship in the Neo4j graph between the clone profile and its connected nodes. Each connected node is notified about the existence of the clone profile, maintaining the integrity of the network.

Output: All affected neighbors of the clone profile are informed, preserving trust within the network.

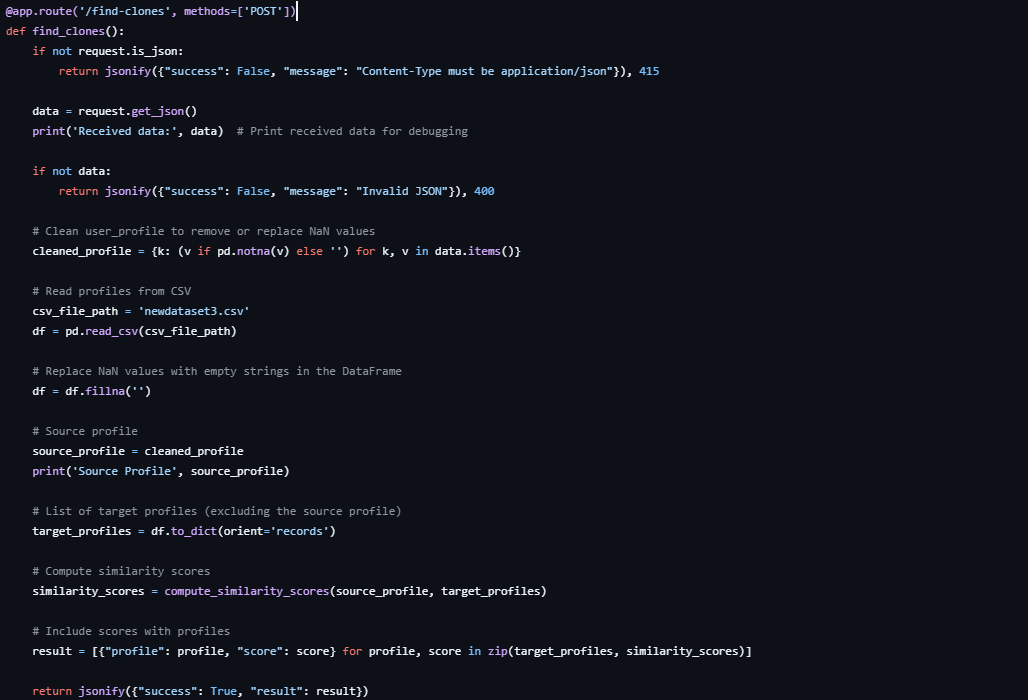
**Implementation Details:**

**Code snippets:**

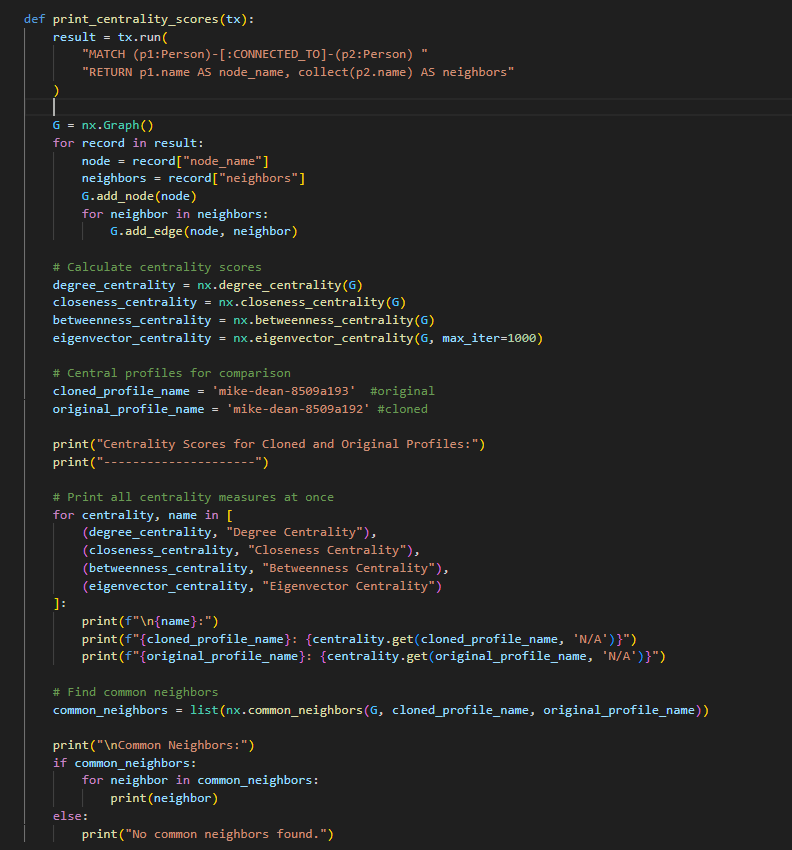
**Python function to calculate similarity between profiles:**

****

**Python function to find clone:**

****

**Python function to find centrality scores and common neighbors of both original and cloned profile.**

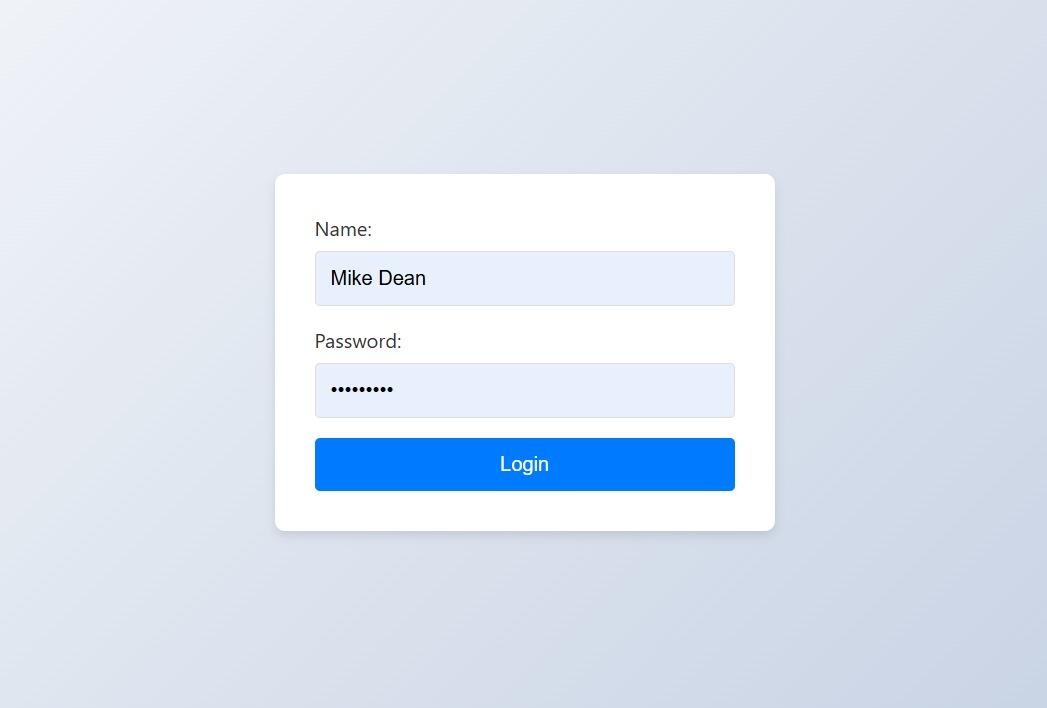
****

**Results and Discussion:**

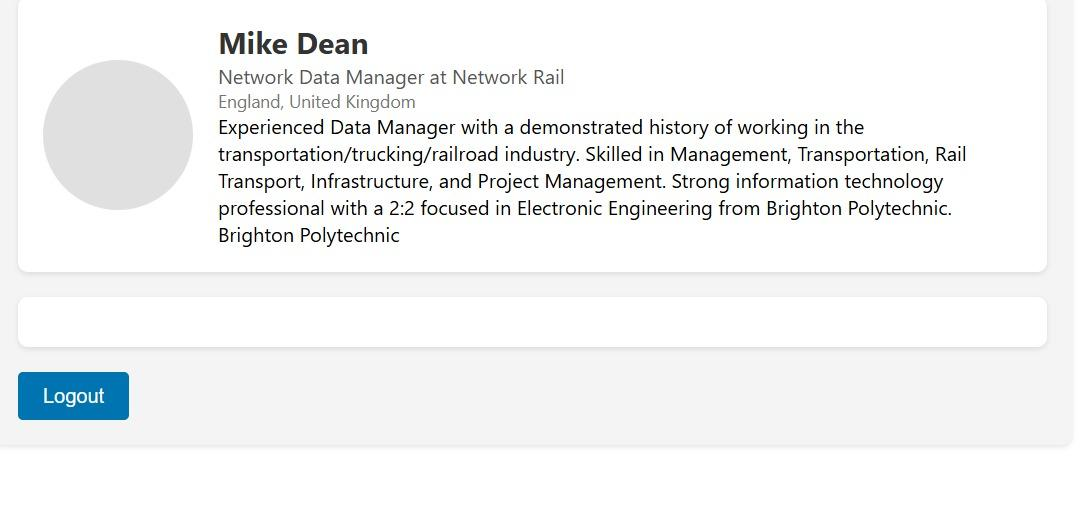
The result of our project will be a list of flagged clones and the assessment results of the impact caused by clones in the network graph. These results will be displayed on the website.

**Output screenshots:**

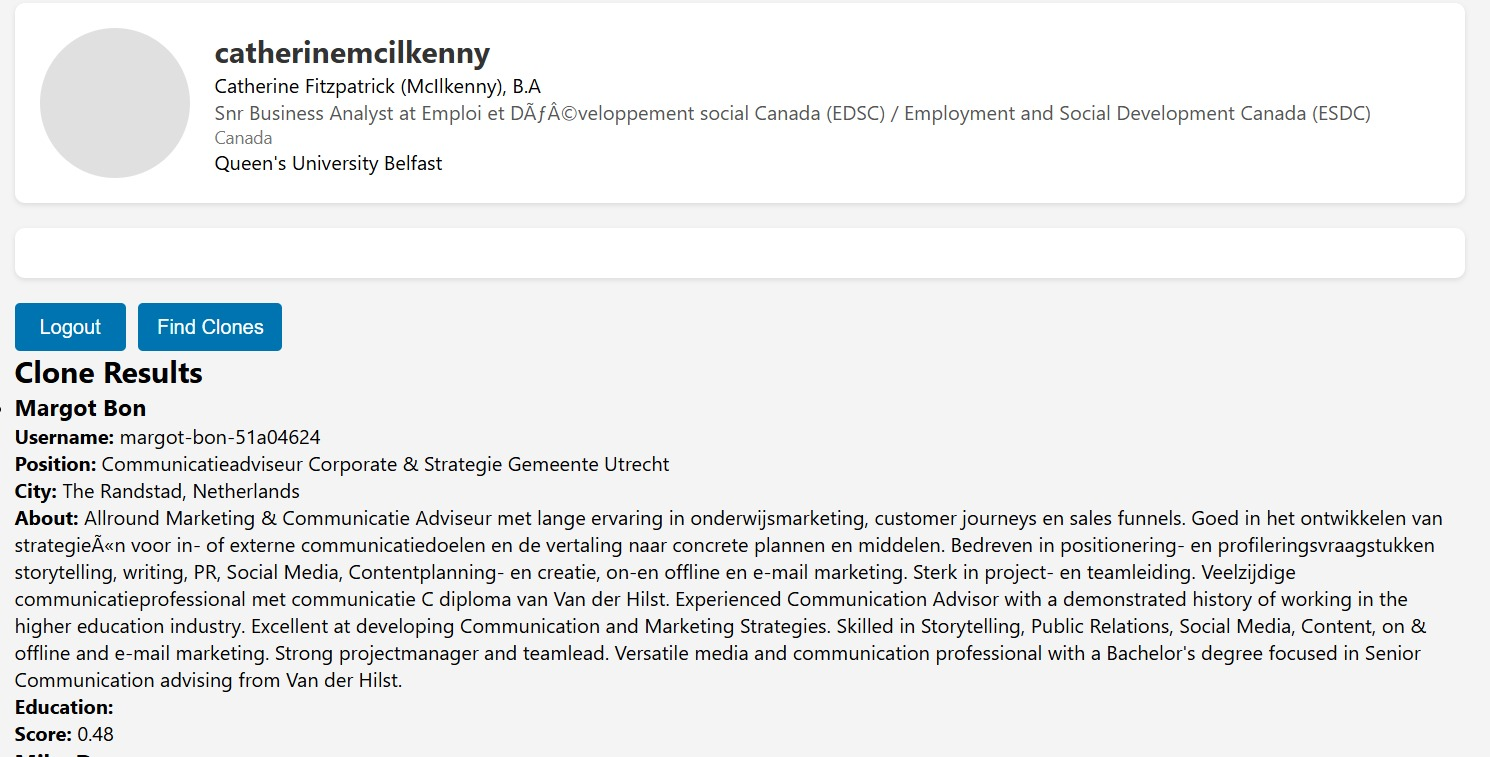
**Login page:**

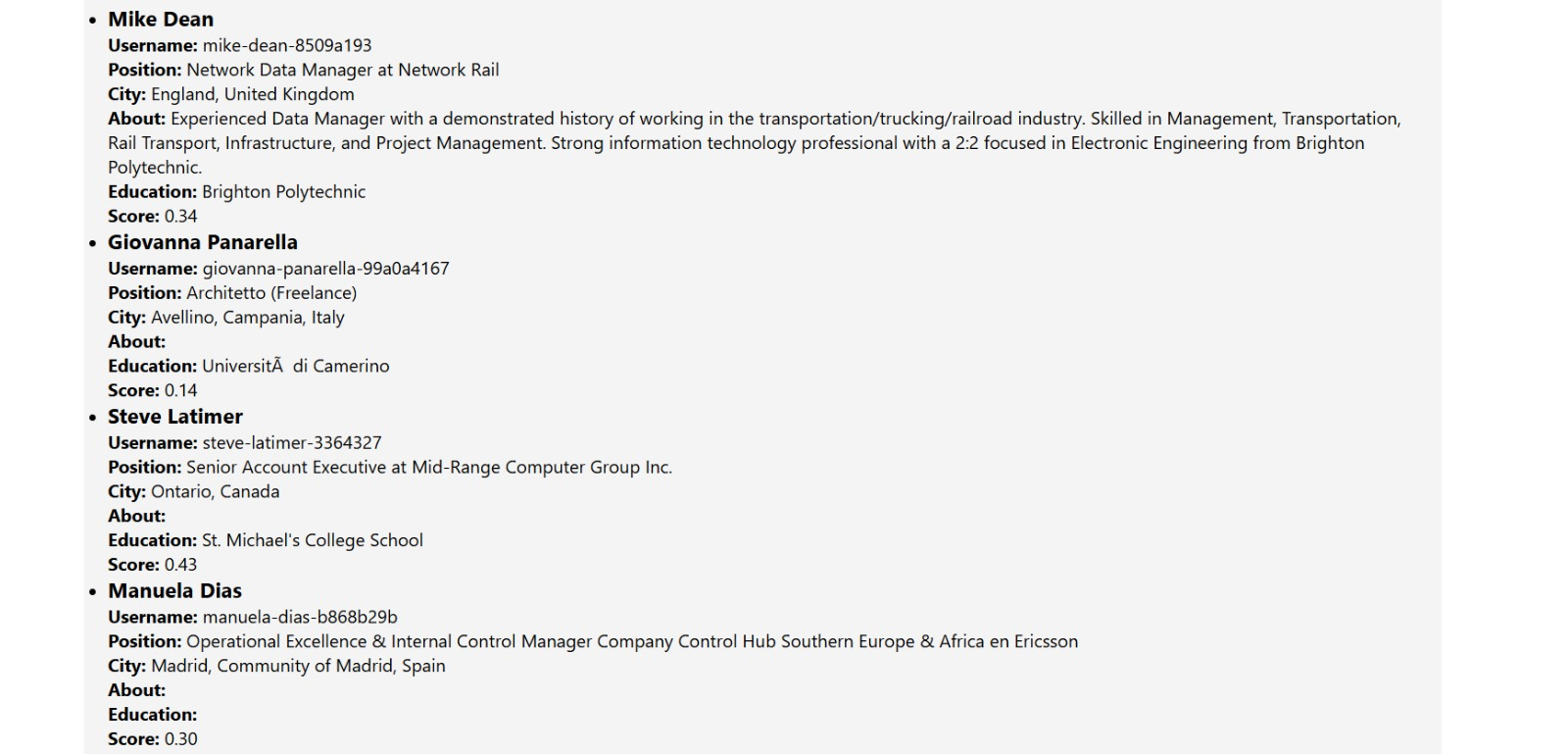
****

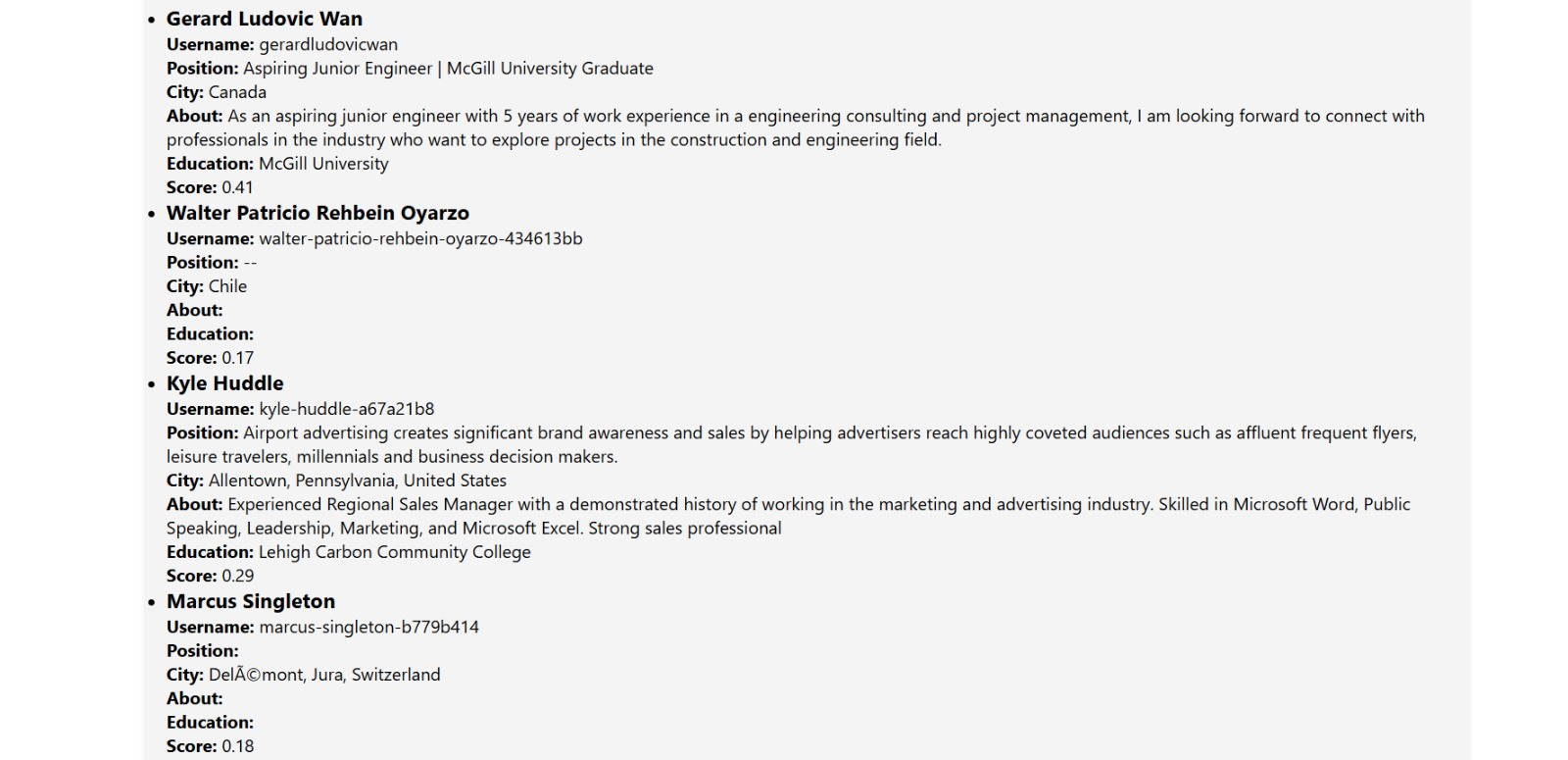
**Profile Page:**

****

**Profile List in Profile Page along with similarity score:**

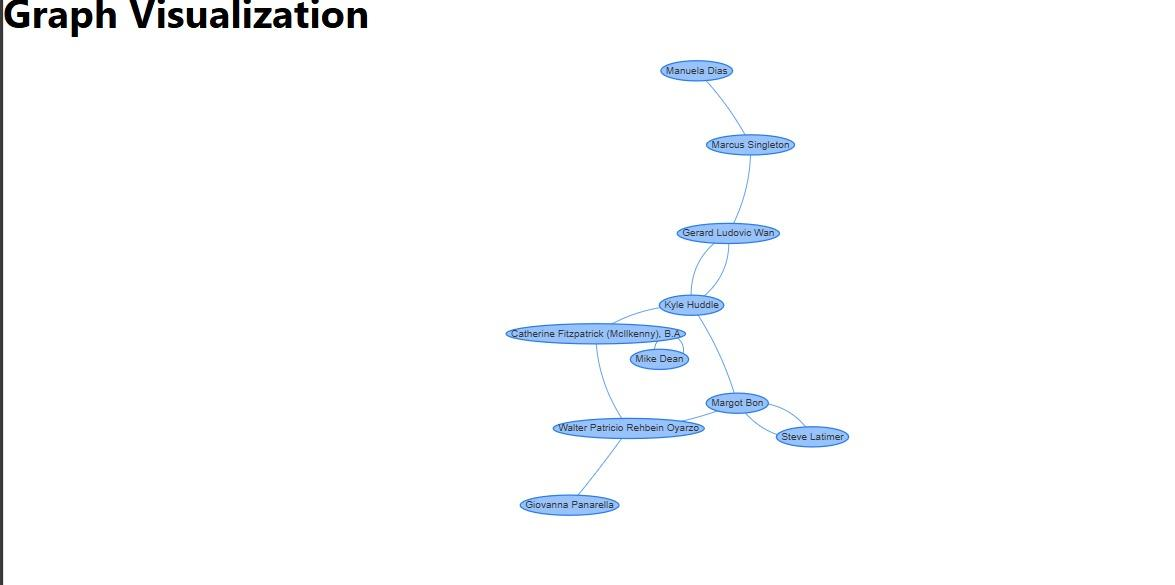
****

****

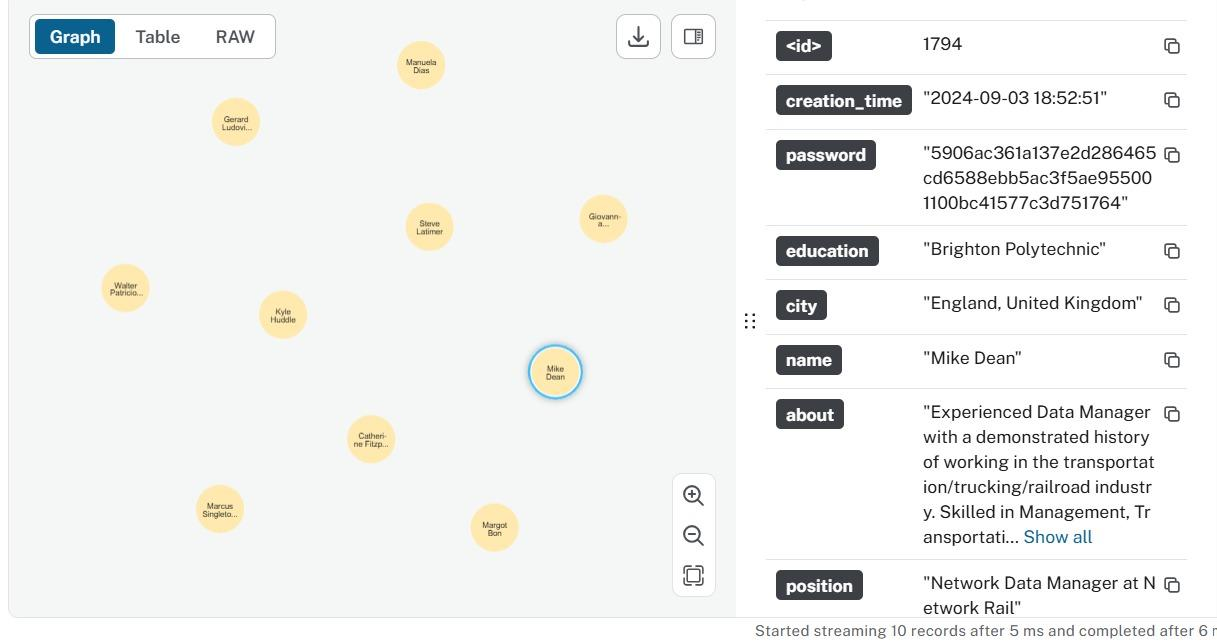
****

*\*Similarity score above 0.78 is considered a clone.*

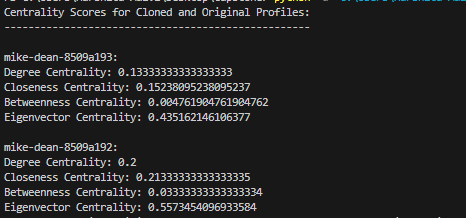
**Graph Visualization in Admin Page:**

****

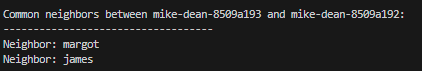
**Node details in Neo4j**

****

**Different centrality scores computed for clone profile and original profile:**



**Common neighbors found for both clone profile and original profile:**



**Notifying all neighbors of the clone profile about the clone:**



Future Work:

**Novel Aspects**:

1. Combination of Sentence Transformers and Levenshtein Algorithm: By using a hybrid approach of semantic string matching and exact string matching, this project offers a novel method of clone profile detection, combining the strengths of both deep learning and traditional algorithms.
2. Centrality-Based Impact Analysis: Utilizing graph centrality metrics (degree, betweenness, closeness, and eigenvector centrality) to assess the influence and impact of a clone profile in the network provides a unique approach to understanding the potential harm caused by clones.
3. Clone-Neighbor Notification Mechanism: The implementation of a system that not only detects clone profiles but also actively informs the network of the clone's existence through a clone-neighbor relationship is an innovative way of maintaining network integrity.
4. Graph Database (Neo4j) for Social Network Simulation: Using Neo4j for simulating LinkedIn-like networks is a unique choice that allows efficient handling of relationships between profiles, facilitating real-time clone detection and network analysis.

**References:**

[1] Sandell, Johan, Einar Asplund, Workneh Yilma Ayele, and Martin Duneld.

"Performance Comparison Analysis of ArangoDB, MySQL, and Neo4j: An Experimental Study of Querying Connected Data." arXiv preprint arXiv:2401.17482 (2024).

[2] Kontaxis, Georgios, Iasonas Polakis, Sotiris Ioannidis, and Evangelos P. Markatos.

"Detecting social network profile cloning." In 2011 IEEE international conference on

pervasive computing and communications workshops (PERCOM Workshops), pp.

295-300. IEEE, 2011. - result: They use syntax based checking, direct string matching, no impact analysis done

[3] Bharti, Nasib Singh Gill, and Preeti Gulia. "Exploring machine learning techniques for fake profile detection in online social networks." Int. J. Electr. Comput. Eng. IJECE 13, no. 3 (2023): 2962.