SCHOOL OF SCIENCE & ENGINEERING

SPRING 2024

CSC 5356 01 Data Engineering and Visualization

Project#2 Recommendation Pipeline

April 1st, 2024

Realized by:

Khadija Salih Alj
Noura Ogbi

Supervised by:

Prof. Tajjedine Rachidi

Table of Contents

I. Project demo and codebase

Link: https://youtu.be/4Cxi3a4jylY

II. Project Team

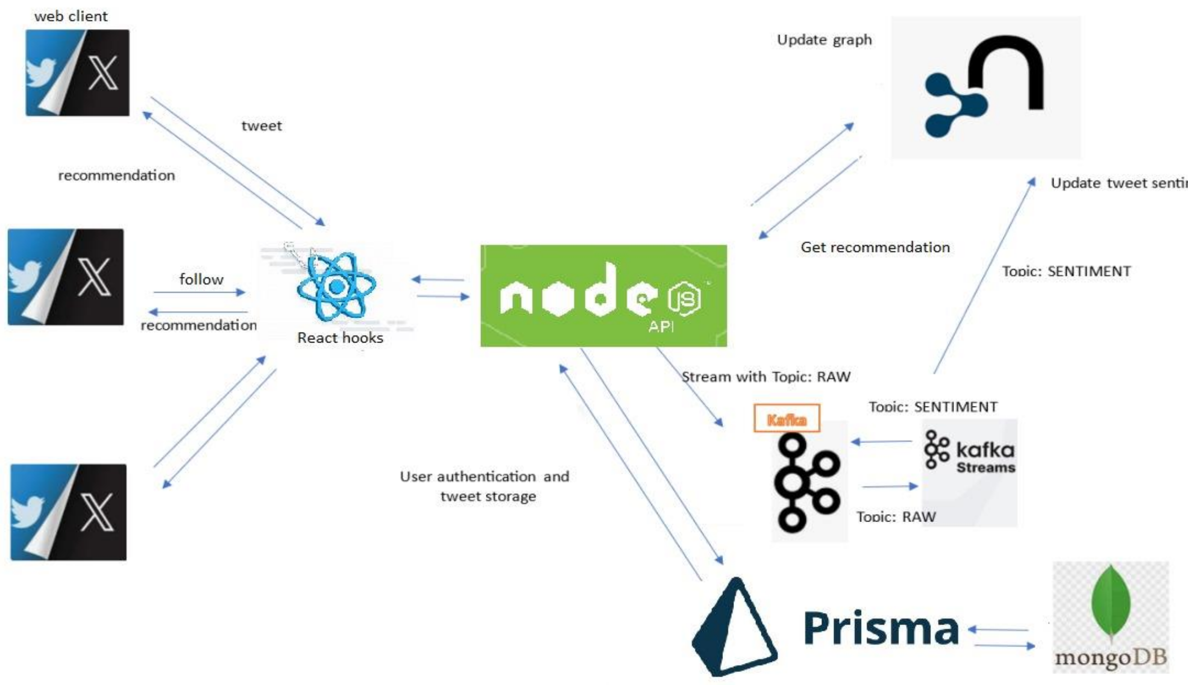
Task		Team member
Front-End Development	Implementing React components for displaying tweets and recommendations.	Khadija
	Managing state with React hooks (implemented in TypeScript).	Khadija & Noura
	Designing and styling user interface.	Noura
Back-End Development	**Database Setup**: Deploy Neo4j database and load twitter graph database.	Noura
	Query Design: Write insertion, recommendation, and sentiment update queries in Cipher.	Khadija & Noura
	MongoDB Schema Design.	Khadija
	Kafka Integration: Design pipeline to stream tweet into "RAW_TWEETS_TOPIC", read from it to perform sentiment analysis, and stream the results into "SENTIMENT_TOPIC", then use that to update the appropriate nodes in Neo4j graph-database.	Khadija & Noura
	Sentiment Analysis Library: Use TextBlob NLP library that is compatible with Kafka Streams for sentiment analysis of tweets streamed through Kafka.	Khadija

III. Project Definition

The project aims to develop a graph-based recommendation system for Twitter/X-like users, enabling personalized recommendations in the form of users to follow. The system leverages a front-end web application for user interaction, a Neo4j graph database for data storage, and NLP processing for sentiment analysis using Kafka Streams. The primary goal is to establish an efficient data processing framework to process tweets, update the graph database, and provide recommendations based on user interactions while ensuring efficient data storage and real-time sentiment analysis.

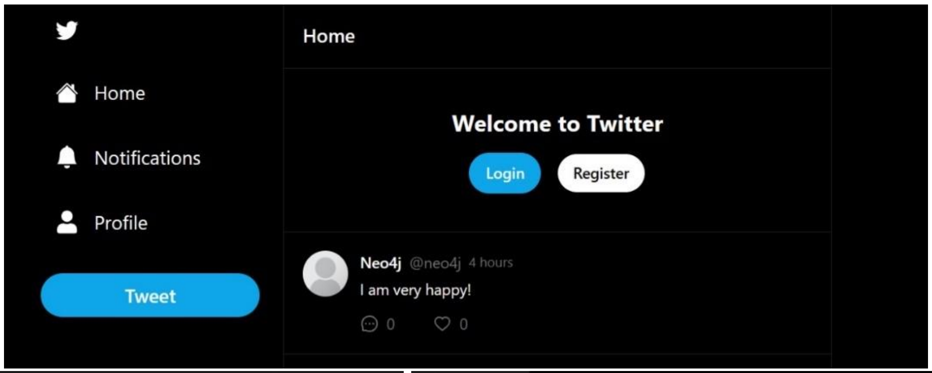
The proposed solution involves the deployment of a Neo4j graph database, the design of a MongoDB schema for storing user profiles and tweets (and more since we went further 😉), and the implementation of sentiment analysis using TextBlob NLP library that works well with the selected stream processing engine, namely Kafka Streams. The sentiment analysis results are streamed using Kafka Message broker and written into the "SENTIMENT_TOPIC" Kafka topic, which is then used to update the appropriate nodes in the Neo4j graph database.

IV. Project Design & Implementation

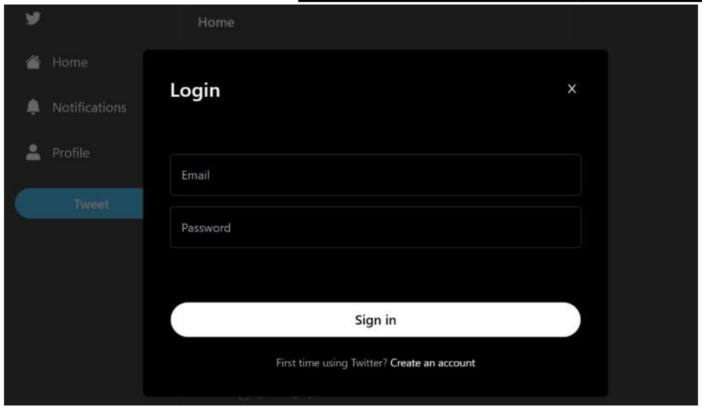


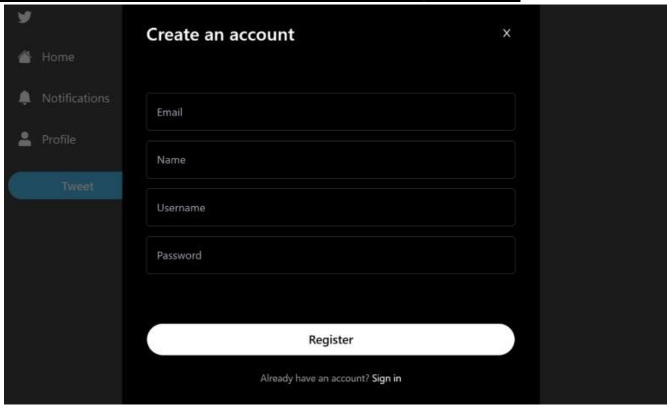
The project is designed to encompass the following key components:

- **Front-End Web Application**: This is where users can authenticate to post tweets and engage with personalized recommendations through an easy-to-use interface. The front-end part of the web application is developed using React for an interactive user experience. Additionally, the back end is powered by Node.js (implemented in TypeScript) to ensure efficient handling of data and seamless integration with the Neo4j graph database and NLP processing components.
    - ➢ Authentication Implementation: Please refer to pages/api/auth/[…nextauth].ts

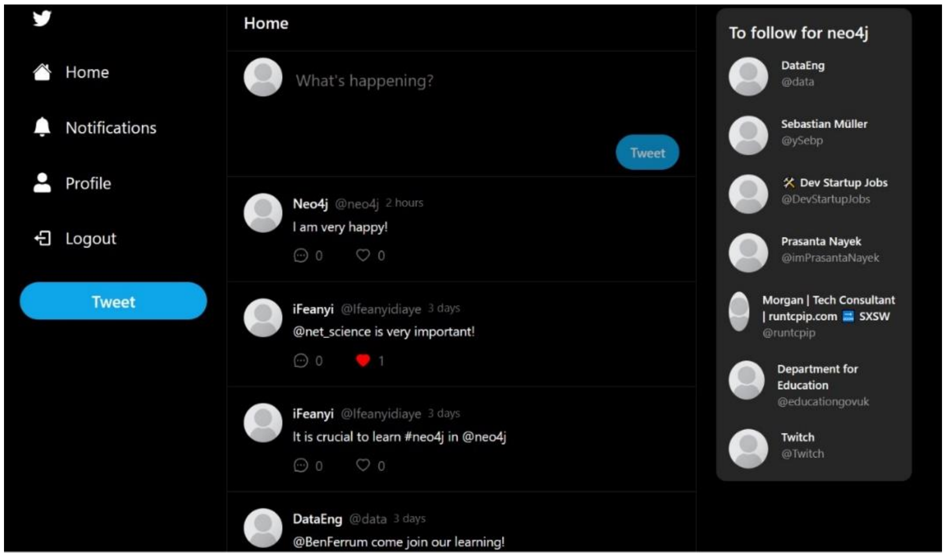


Neo4j @neo4j 4 hours
I am very happy!
0 0





- ➢ Components Implementation: Please refer to the components folder in our code.
- ➢ React hooks: Please refer to the hooks folder in our code.
- ➢ Home Page:



Neo4j @neo4j 2 hours
I am very happy!
0 0

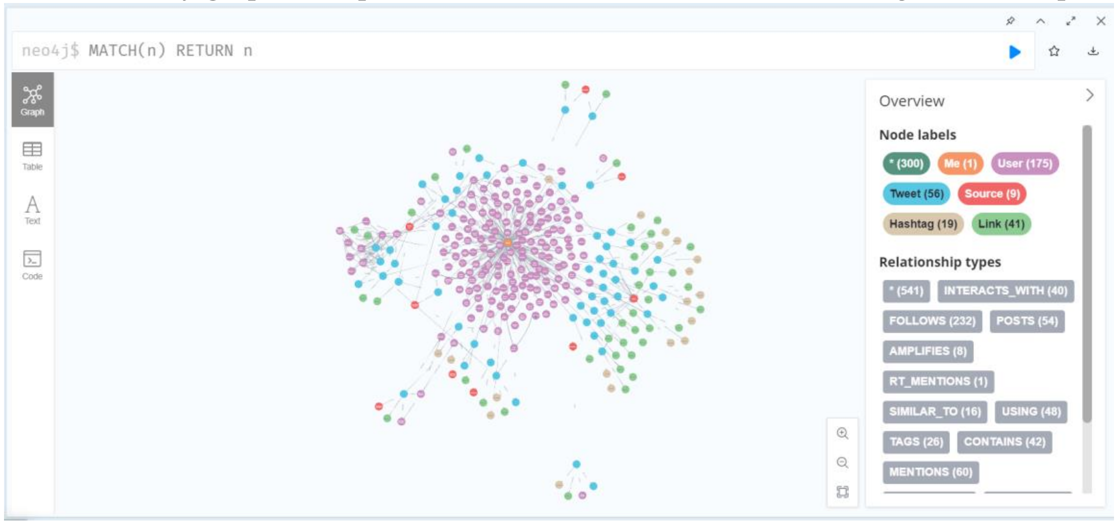
iFeanyi @Ifeanyidiaye 3 days
@net_science is very important!
0 1

iFeanyi @Ifeanyidiaye 3 days
It is crucial to learn #neo4j in @neo4j
0 0

DataEng @data 3 days
@BenFerrum come join our learning!

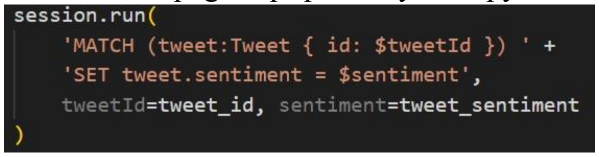
DataEng @data
Sebastian Müller @ySebp
🛠 Dev Startup Jobs @DevStartupJobs
Prasanta Nayek @imPrasantaNayek
Morgan | Tech Consultant | runtcpip.com SXSW @runtcpip
Department for Education @educationgovuk
Twitch @Twitch

- **Neo4j Graph Database**: The Neo4j graph database is designed to represent the relationships between users, tweets, hashtags, and their interactions. It utilizes a graph-based data model consisting of nodes and relationships to capture the complex interconnections within the Twitter-like platform. The nodes represent entities such as users, tweets, and hashtags, while the relationships capture the connections and interactions between these entities (like FOLLOWS, POSTS, MENTIONS, TAGS). This model enables efficient querying and traversal of the graph structure to derive meaningful insights and recommendations based on user interactions and sentiments. Additionally, as part of the project setup, we created a local DBMS where we cloned the data from *neo4j-graph-examples/twitter-v2* database to facilitate testing and development.



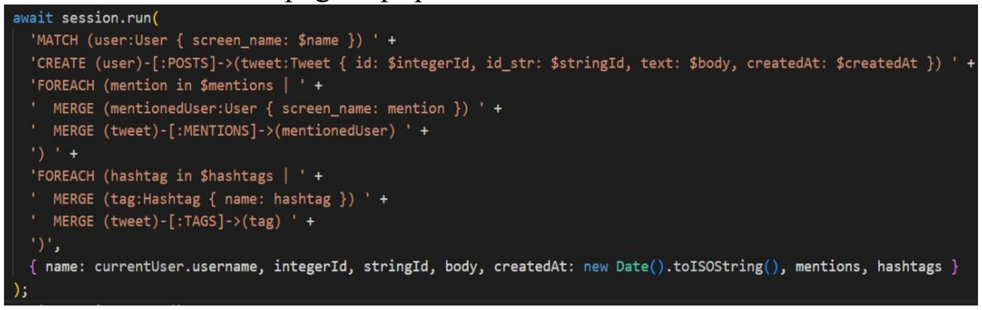
```
neo4j$ MATCH(n) RETURN n
```

  - ➢ Update Sentiment: Please refer to pages/api/posts/try/start.py



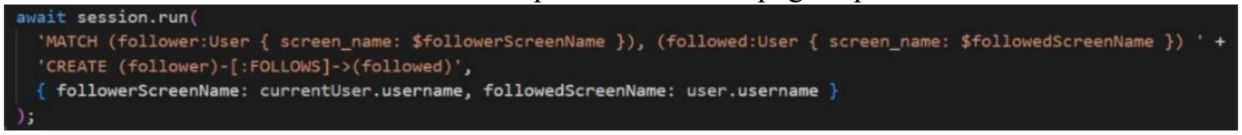
```
session.run(
    'MATCH (tweet:Tweet { id: $tweetId }) ' +
    'SET tweet.sentiment = $sentiment',
    tweetId=tweet_id, sentiment=tweet_sentiment
)
```

  - ➢ Insert tweet: Please refer to pages/api/posts/index.ts



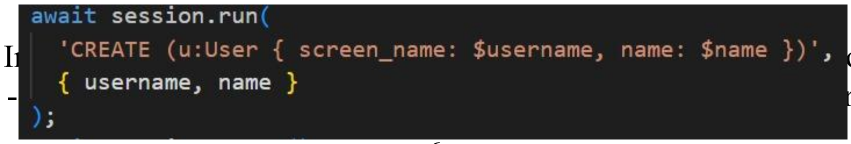
```
await session.run(
  'MATCH (user:User { screen_name: $name }) ' +
  'CREATE (user)-[:POSTS]->(tweet:Tweet { id: $integerId, id_str: $stringId, text: $body, createdAt: $createdAt }) ' +
  'FOREACH (mention in $mentions | ' +
  '  MERGE (mentionedUser:User { screen_name: mention }) ' +
  '  MERGE (tweet)-[:MENTIONS]->(mentionedUser) ' +
  ') ' +
  'FOREACH (hashtag in $hashtags | ' +
  '  MERGE (tag:Hashtag { name: hashtag }) ' +
  '  MERGE (tweet)-[:TAGS]->(tag) ' +
  ')',
  { name: currentUser.username, integerId, stringId, body, createdAt: new Date().toISOString(), mentions, hashtags }
);
```

  - ➢ Create FOLLOWS relationship: Please refer to pages/api/follow.ts



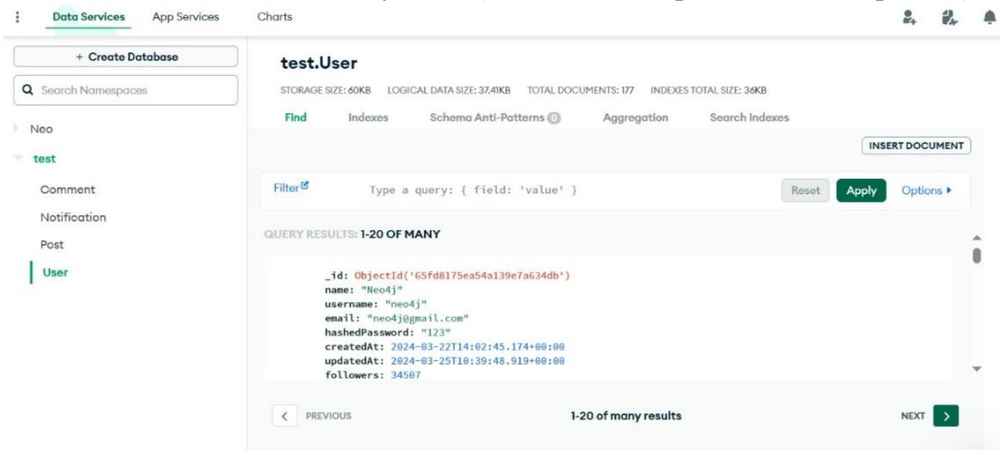
```
await session.run(
  'MATCH (follower:User { screen_name: $followerScreenName }), (followed:User { screen_name: $followedScreenName }) ' +
  'CREATE (follower)-[:FOLLOWS]->(followed)',
  { followerScreenName: currentUser.username, followedScreenName: user.username }
);
```

  - ➢ Insert User: Please refer to pages/api/register.ts



```
await session.run(
  'CREATE (u:User { screen_name: $username, name: $name })',
  { username, name }
);
```

- **MongoDB**: I ofiles and tweets. The schema - ributes associated with user profiles, such as user information, and social interactions. Similarly, the schema for tweets encompasses relevant metadata, and content. This schema design ensures the efficient storage and retrieval of user-centric and tweet-related data, contributing to the overall effectiveness of the recommendation system. (Please refer to prisma/schema.prisma)



test.User

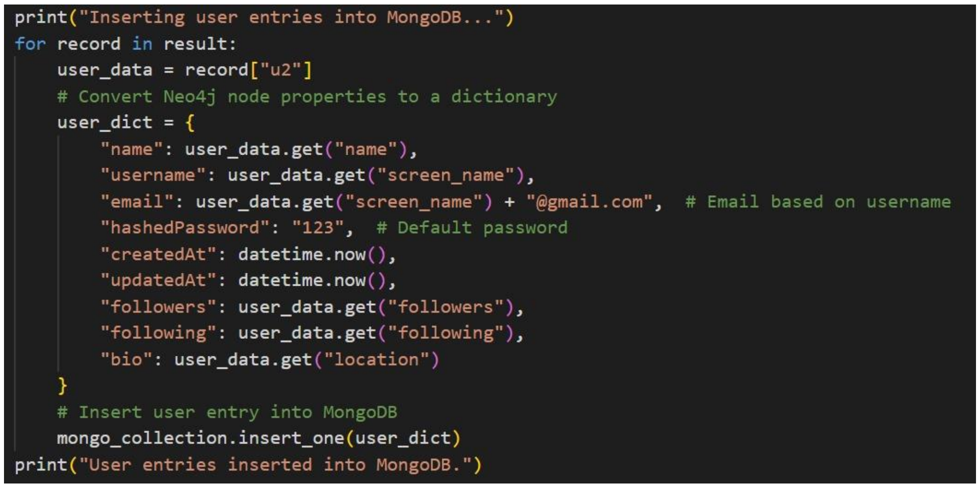
STORAGE SIZE: 60KB LOGICAL DATA SIZE: 37.41KB TOTAL DOCUMENTS: 177 INDEXES TOTAL SIZE: 36KB

Type a query: { field: 'value' }

QUERY RESULTS: 1-20 OF MANY

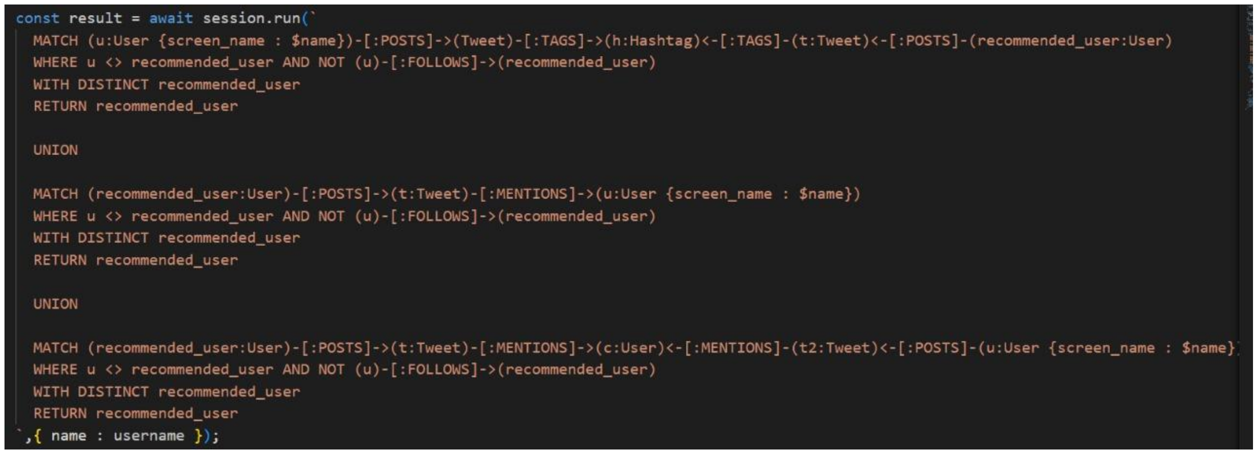
```
_id: ObjectId('65fd8175ea54a139e7a634db')
name: "Neo4j"
username: "neo4j"
email: "neo4j@gmail.com"
hashedPassword: "123"
createdAt: 2024-03-22T14:02:45.174+00:00
updatedAt: 2024-03-25T10:39:48.919+00:00
followers: 34507
```

1-20 of many results



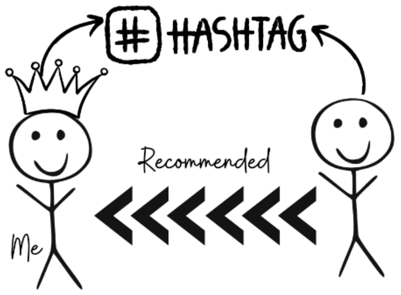
```python
print("Inserting user entries into MongoDB...")
for record in result:
    user_data = record["u2"]
    # Convert Neo4j node properties to a dictionary
    user_dict = {
        "name": user_data.get("name"),
        "username": user_data.get("screen_name"),
        "email": user_data.get("screen_name") + "@gmail.com",  # Email based on username
        "hashedPassword": "123",  # Default password
        "createdAt": datetime.now(),
        "updatedAt": datetime.now(),
        "followers": user_data.get("followers"),
        "following": user_data.get("following"),
        "bio": user_data.get("location")
    }
    # Insert user entry into MongoDB
    mongo_collection.insert_one(user_dict)
print("User entries inserted into MongoDB.")
```

- **Recommendation System**: Utilizes Cipher queries to provide personalized recommendations based on user tweets, tags and hashtags, sentiments, and interactions. (Please refer to pages/api/users/index.ts)

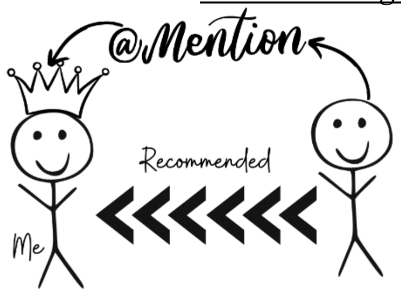


```
const result = await session.run(`
  MATCH (u:User {screen_name : $name})-[:POSTS]->(Tweet)-[:TAGS]->(h:Hashtag)<-[:TAGS]-(t:Tweet)<-[:POSTS]-(recommended_user:User)
  WHERE u <> recommended_user AND NOT (u)-[:FOLLOWS]->(recommended_user)
  WITH DISTINCT recommended_user
  RETURN recommended_user

  UNION

  MATCH (recommended_user:User)-[:POSTS]->(t:Tweet)-[:MENTIONS]->(u:User {screen_name : $name})
  WHERE u <> recommended_user AND NOT (u)-[:FOLLOWS]->(recommended_user)
  WITH DISTINCT recommended_user
  RETURN recommended_user

  UNION

  MATCH (recommended_user:User)-[:POSTS]->(t:Tweet)-[:MENTIONS]->(c:User)<-[:MENTIONS]-(t2:Tweet)<-[:POSTS]-(u:User {screen_name : $name}
  WHERE u <> recommended_user AND NOT (u)-[:FOLLOWS]->(recommended_user)
  WITH DISTINCT recommended_user
  RETURN recommended_user
`,{ name : username });
```

- ✓ It starts by matching the user (designated by $name – knowing that the username is unique to each user) who has posted tweets tagging the same hashtags, but whom the original user does not already follow. Additionally, the DISTINCT keyword ensures that each recommended user is returned only once.



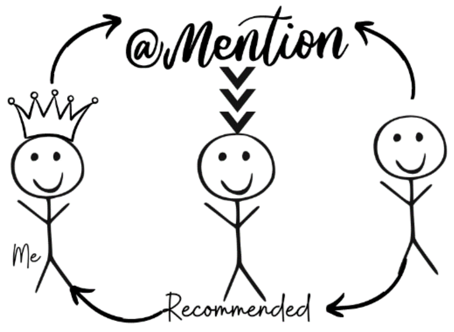
➢ *Recommending users who mentioned the concerned one:*



- ✓ This part returns users who have mentioned the original user in their tweets. Similar to the first part, it identifies users whom the original user does not already follow.

➢ *Recommending users who mentioned same user as done by concerned one:*

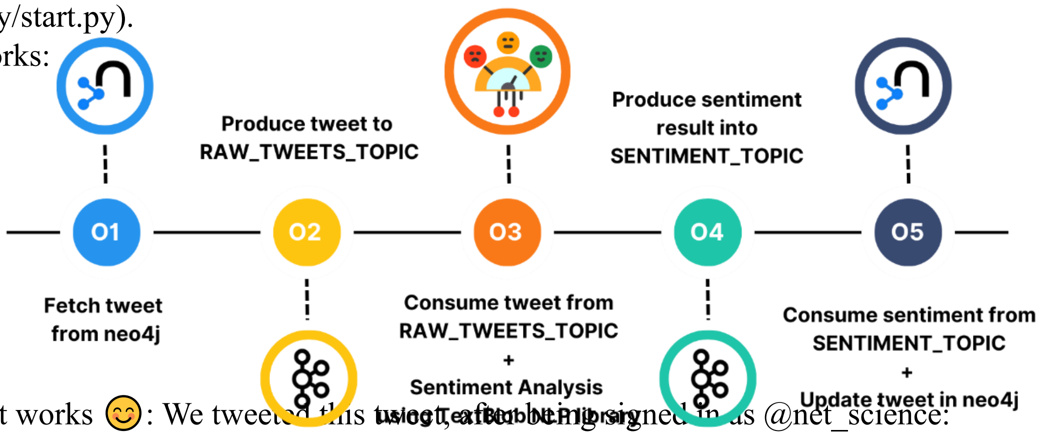
- ✓ This part matches users who are mentioning the same user; It looks for tweets posted by the recommended user (designated by recommended_user) that mention another user (designated by c) and finds tweets by other users (designated by u) that also mention the same user (c).



Combining Recommendations:

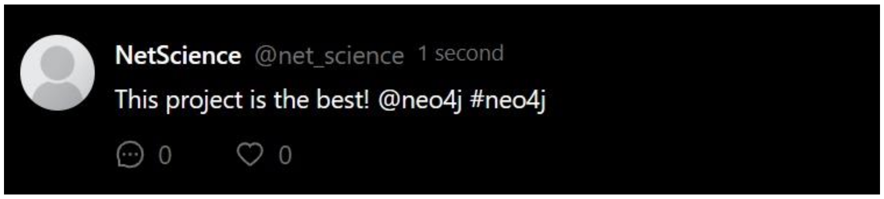
The results from the first, second, and third parts are combined using the UNION operator to ensure that the final list of recommended users includes both those who share similar interests (hashtags), those who have mentioned the original user directly, and who have mentioned the same user.

- **Kafka Message Broker**: This component streams raw tweets for sentiment analysis and efficiently updates the graph database with sentiment information. (Please refer to pages/api/posts/try/start.py).
  - ➢ How it works:



Produce tweet to RAW_TWEETS_TOPIC
Produce sentiment result into SENTIMENT_TOPIC
01
02
03
04
05
Fetch tweet from neo4j
Consume tweet from RAW_TWEETS_TOPIC + Sentiment Analysis
Consume sentiment from SENTIMENT_TOPIC + Update tweet in neo4j

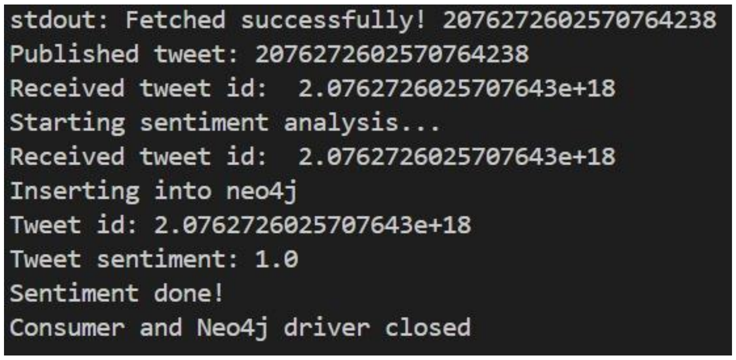
  - ➢ Showing it works 😊: We tweeted this tweet, after being signed in as @net_science:


NetScience @net_science 1 second

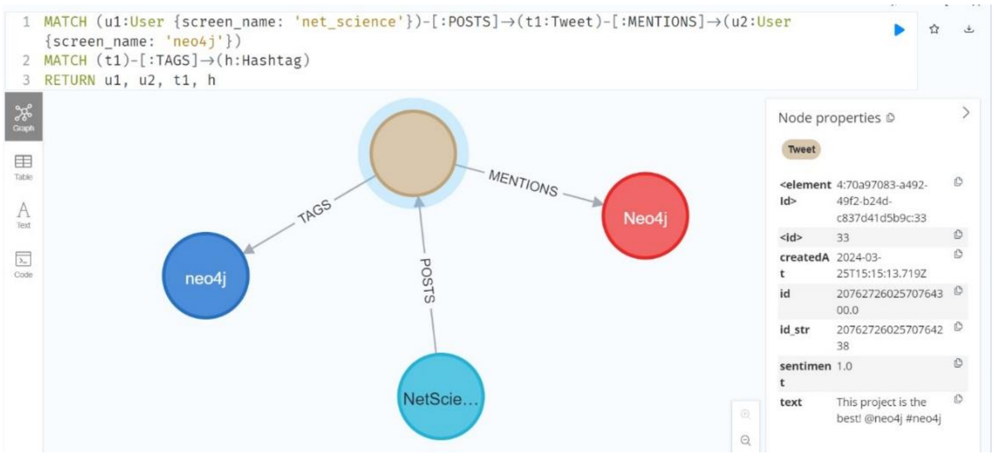
This project is the best! @neo4j #neo4j

0 0

- Output in terminal:


```
stdout: Fetched successfully! 2076272602570764238
Published tweet: 2076272602570764238
Received tweet id:  2.0762726025707643e+18
Starting sentiment analysis...
Received tweet id:  2.0762726025707643e+18
Inserting into neo4j
Tweet id: 2.0762726025707643e+18
Tweet sentiment: 1.0
Sentiment done!
Consumer and Neo4j driver closed
```

- Neo4j proof:

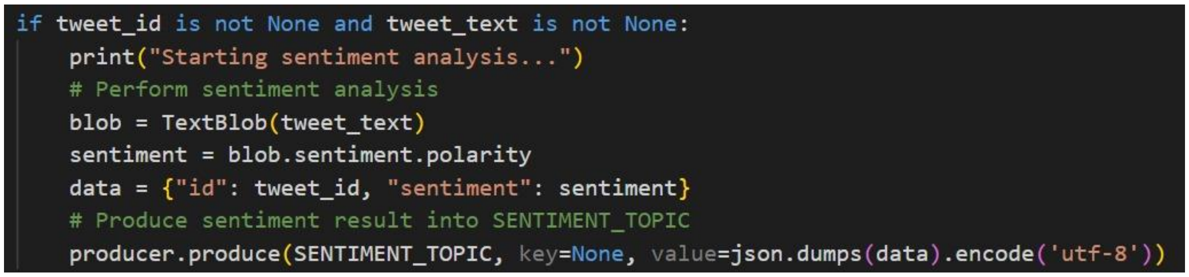

```
MATCH (u1:User {screen_name: 'net_science'})-[:POSTS]→(t1:Tweet)-[:MENTIONS]→(u2:User
{screen_name: 'neo4j'})
MATCH (t1)-[:TAGS]→(h:Hashtag)
RETURN u1, u2, t1, h
```

Node properties

Tweet

<element Id>	4:70a97083-a492-49f2-b24d-c837d41d5b9c:33
<id>	33
createdAt	2024-03-25T15:15:13.719Z
id	207627260257076430 0.0
id_str	2076272602570764238
sentiment	1.0
text	This project is the best! @neo4j #neo4j

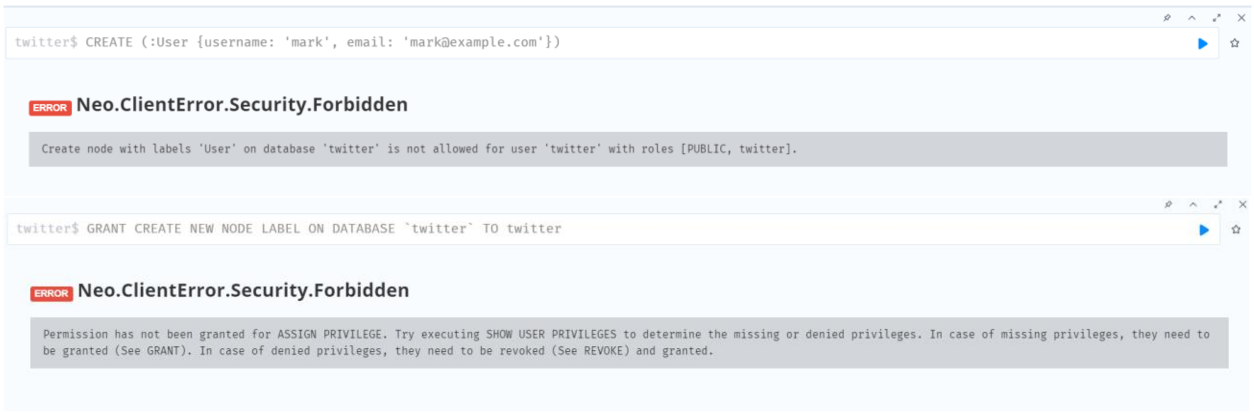
- **NLP Processing Engine**: This component utilizes the Kafka message broker and the **TextBlob** library for sentiment analysis. The sentiment analysis categorizes tweets as either having a negative sentiment (-1), neutral sentiment (0), or positive sentiment (+1). Additionally, the neutral, represented as a float ranging from 0 to 1, is also evaluated.


```python
if tweet_id is not None and tweet_text is not None:
    print("Starting sentiment analysis...")
    # Perform sentiment analysis
    blob = TextBlob(tweet_text)
    sentiment = blob.sentiment.polarity
    data = {"id": tweet_id, "sentiment": sentiment}
    # Produce sentiment result into SENTIMENT_TOPIC
    producer.produce(SENTIMENT_TOPIC, key=None, value=json.dumps(data).encode('utf-8'))
```

V. Implementation Limitations

During the implementation phase, the following limitations were encountered, along with their corresponding resolutions:

- **Dataset Modification Restrictions**: Initially, difficulties arose in modifying (inserting, creating) entries in the Neo4j default twitter dataset due to permission constraints. To address this, a local DBMS was created in Neo4j, and the data was cloned from the neo4j-graph-examples/twitter-v2 database.



```
twitter$ CREATE (:User {username: 'mark', email: 'mark@example.com'})
```

ERROR Neo.ClientError.Security.Forbidden

```
Create node with labels 'User' on database 'twitter' is not allowed for user 'twitter' with roles [PUBLIC, twitter].
```

```
twitter$ GRANT CREATE NEW NODE LABEL ON DATABASE `twitter` TO twitter
```

ERROR Neo.ClientError.Security.Forbidden

```
Permission has not been granted for ASSIGN PRIVILEGE. Try executing SHOW USER PRIVILEGES to determine the missing or denied privileges. In case of missing privileges, they need to
be granted (See GRANT). In case of denied privileges, they need to be revoked (See REVOKE) and granted.
```

- **Sentiment Analysis Libraries**: Subsequently, challenges were faced in selecting suitable sentiment analysis libraries. Ultimately, the TextBlob library was identified as the most suitable solution.
- **Scalability, Automation, and Stream Processing**: Efforts were focused on improving the scalability, automation, and stream processing capabilities to enhance the overall system performance and efficiency.

VI. Final Note

The journey of building our project has been incredibly rewarding, starting as a daunting challenge, and evolving into a fulfilling experience. With grit and determination, we pushed through late nights and tough bugs, inching closer to our goal each day. Despite the setbacks, we persevered, and our hard work paid off as we completed the project one week ahead of schedule. This journey has been a testament to our dedication and passion for learning. We have grown immensely, both as individuals and as a team, and this project will forever hold a special place in our hearts as a symbol of our perseverance and triumph. Thank you infinitely, dear Professor, Rachidi!