EX.NO: 01	DESIGN OWN SOCIAL MEDIA APPLICATION
DATE:	

AIM:

To implement social media application.

ALGORITHM:

STEP 1: Create a new directory for your project. Inside this directory, create the following subdirectories and files.

STEP 2: Open a terminal and navigate to your project directory.

STEP 3: Flask : the web framework used for building the web application. render_template : A function from Flask that renders HTML templates. Graph, Namespace, Literal, URIRef : These are classes from the rdflib library, used for working with RDF (Resource Description Framework). RDF is a framework for representing information about resources on the web.

STEP 4: create an instance of the Flask class, representing the web application

STEP 5: social_graph: An instance of the RDF Graph used to store social data. FOAF: A Namespace object representing the Friend of a Friend (FOAF) vocabulary. FOAF is commonly used for describing people and relationships on the web.

STEP 6: URIRef: Represents a URI reference. Sample user data is added to the RDF graph, including user URIs and their names.

STEP 7: Adds a friendship relationship between user1 and user2 in the RDF graph.

STEP 8: Defines a route for the root URL (/). When a user accesses this URL, the index function is called. The index function retrieves a list of users from the RDF graph and renders the 'index.html' template, passing the users, social graph, and FOAF namespace to the template.

STEP 9: Defines a route for the '/profile/<user_id>' URL pattern. The <user_id> part is a dynamic parameter. The profile function takes the user_id as a parameter, retrieves the user's information from the RDF graph, and renders the 'profile.html' template, passing the user's name, friends, social graph, and FOAF namespace to the template.

STEP 10: Checks if the script is being run directly (not imported as a module). If so, it starts the Flask development server with debugging enabled.

PROGRAM:

index.html:

```
<!-- templates/index.html -->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Social Media App</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>
    <div style="text-align: center;">
        <h1>Users</h1>
    </div>
    <hr>
    <div class="image-container">
        {% for user in users %}
        <a href="{{ url_for('profile', user_id=user.split('/')[-1]) }}">
            <img src="https://tse1.mm.bing.net/th?
id=OIP.eoBtu339Epu84pJA0EY_QwAAAA&pid=Api&P=0&h=180"
                alt="User Image" class="avatar">
            <div class="overlay">{{ social_graph.value(user, FOAF.name) }}</div>
        </a>
        {% endfor %}
    </div>
</body>

</html>
```

profile.html:

```
<!-- templates/profile.html -->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Profile</title>
```

```
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>
    <h1>User Profile</h1>
    <p>Name: {{ user_name }}</p>
    <h2>Friends</h2>
    <ul>
        {% for friend in friends %}
        <li>{{ social_graph.value(friend, FOAF.name) }}</li>
        {% endfor %}
    </ul>
</body>

</html>
```

styles.css:

```
/* static/styles.css */
body {
    font-family: 'Times New Roman', Times, serif;
    margin: 20px;
    background-color: aliceblue;
}

h1, h2 {
    color: #333;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    margin-bottom: 10px;
}

/* Define a basic styling for the image container */
.image-container {
    display: flex;
    justify-content: space-evenly;
    max-width: 800px; /* Adjust the max-width based on your design */
    margin: auto; /* Center the container */
```

```
}

/* Style for each individual image container */
.image-container a {
    position: relative;
    text-decoration: none;
    display: inline-block; /* Ensure block-level layout for the anchor */
}

/* Style for each individual image */
.image-container img {
    width: 100%; /* Set the width to 100% to match the container size */
    height: auto; /* Auto-adjust height to maintain the aspect ratio */
    margin-right: 16px; /* Add some spacing between images */
    transition: transform 0.3s; /* Add a smooth transition effect */
    display: block; /* Ensure block-level layout for the image */
}

/* Style for the text overlay */
.image-container .overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%; /* Set the width to 100% to match the container size */
    height: 100%; /* Set the height to 100% to match the container size */
    display: flex;
    align-items: center;
    justify-content: center;
    opacity: 0;
    background: rgba(0, 0, 0, 0.5); /* Semi-transparent background */
    color: #fff; /* Text color */
    transition: opacity 0.3s; /* Add a smooth transition effect */
    pointer-events: none; /* Ensure the overlay doesn't block interactions with the
underlying image */
}

/* Hover effect on images */
.image-container a:hover .overlay {
    opacity: 1;
}

/* Style for the image links */
.image-container a {
    text-decoration: none; /* Remove underlines from links */
```

```
    color: inherit; /* Inherit text color from the parent */
}
```

app.py:

```
from flask import Flask, render_template, request

from rdflib import Graph, Namespace, Literal, URIRef

app = Flask(__name__)

# RDF graph to store social data
social_graph = Graph()

# Define Namespace
FOAF = Namespace("http://xmlns.com/foaf/0.1/")

# Sample user data
user_data = {
    "1": ("Luffy", ["2", "3", "4"]),
    "2": ("Zoro", ["1", "4", "3"]),
    "3": ("Nami", ["1", "4", "2"]),
    "4": ("Usopp", ["1", "3", "2"])
}

# Populate RDF graph with sample data
for user_id, (name, friends) in user_data.items():
    user_uri = URIRef(f"http://example.com/users/{user_id}")
    social_graph.add((user_uri, FOAF.name, Literal(name)))
    for friend_id in friends:
        friend_uri = URIRef(f"http://example.com/users/{friend_id}")
        social_graph.add((user_uri, FOAF.knows, friend_uri))

@app.route('/')
def index():
    # Display a list of users
    users = social_graph.subjects(predicate=FOAF.name)
    return render_template('index.html', users=users, social_graph=social_graph,
FOAF=FOAF)

@app.route('/profile/<user_id>')
def profile(user_id):
    try:
        user = URIRef(f"http://example.com/users/{user_id}")
```

```
        user_name = social_graph.value(user, FOAF.name)
        friends = social_graph.objects(subject=user, predicate=FOAF.knows)
        return render_template('profile.html', user_name=user_name, friends=friends,
                               social_graph=social_graph, FOAF=FOAF)
    except Exception as e:
        return render_template('error.html', error_message=str(e))

if __name__ == '__main__':
    app.run(debug=True)
```

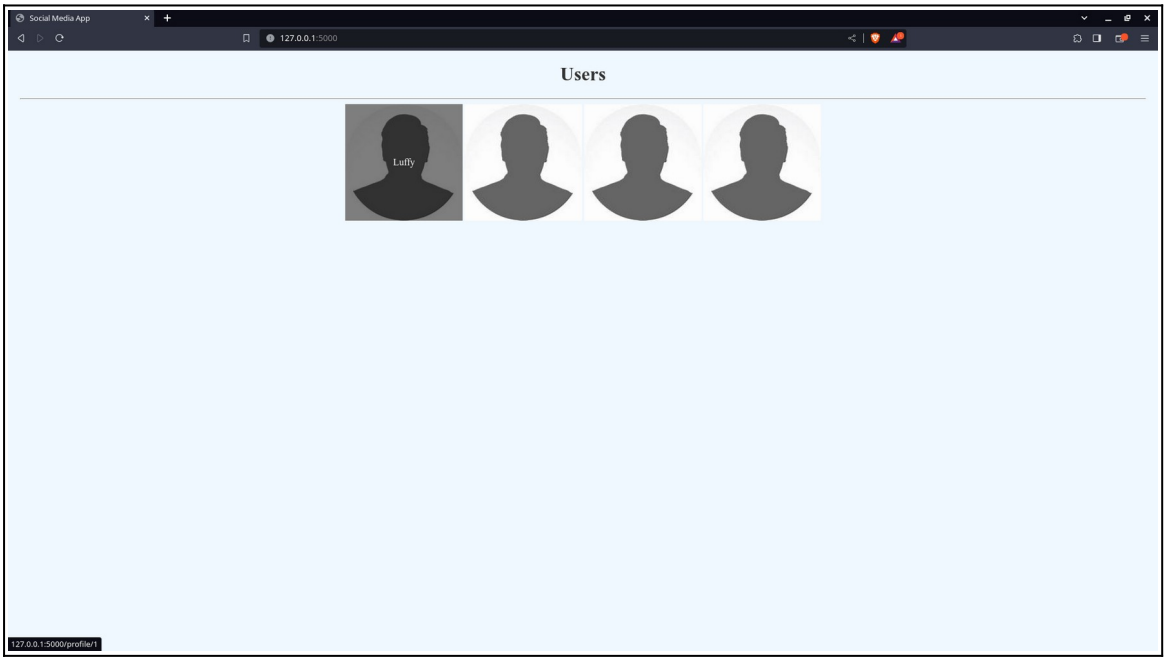
PROJECT STRUCTURE: (Just For Ref.)

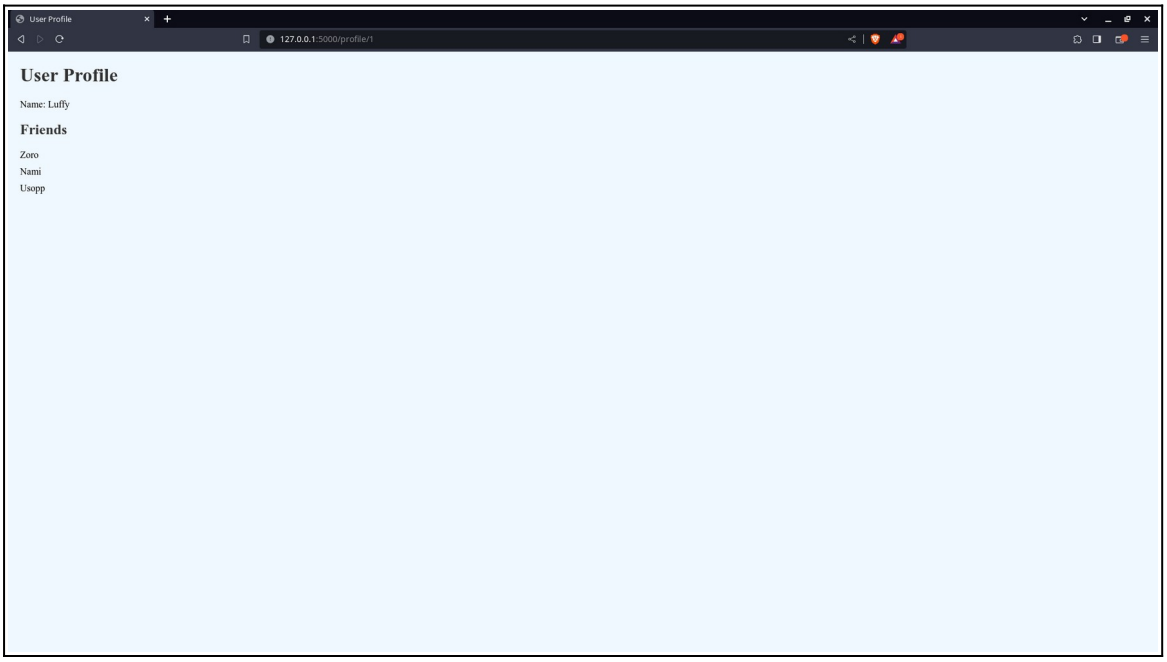
```
.
├── app.py
├── static
│   └── styles.css
└── templates
    ├── index.html
    └── profile.html

2 directories, 4 files
```

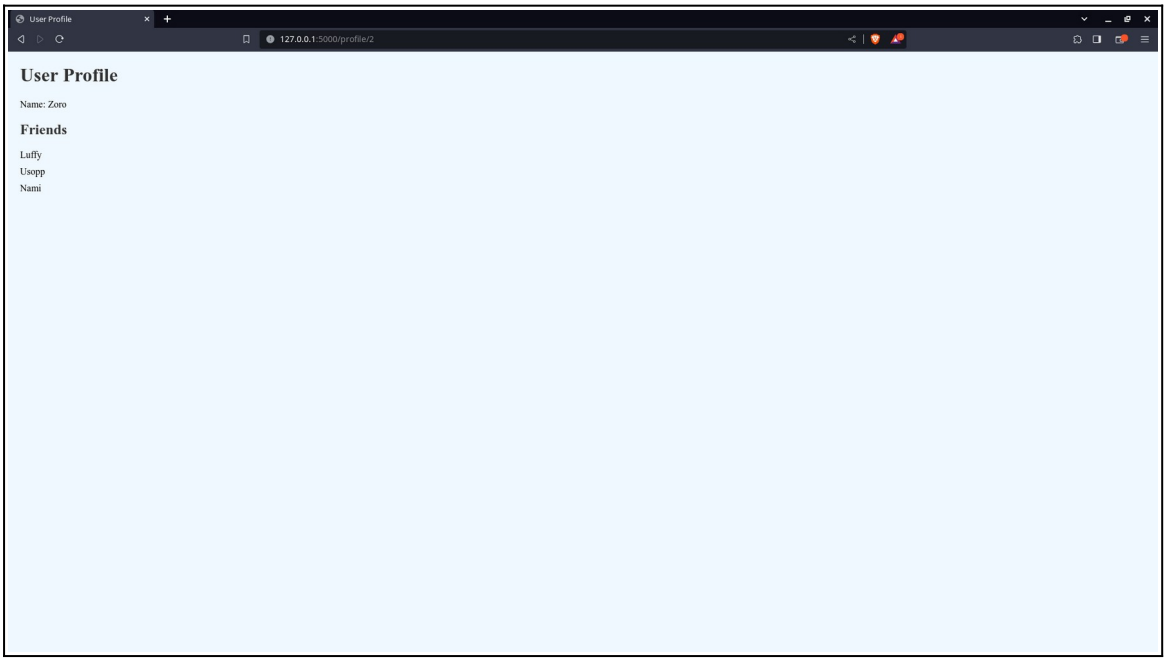
Execution (Python (Programming Language)):

```
$ pip3 install Flask
$ pip3 install rdflib
$ python3 app.py
```

OUTPUT:


Social Media App
127.0.0.1:5000
Users
Luffy
127.0.0.1:5000/profile/1


User Profile
127.0.0.1:5000/profile/1
User Profile
Name: Luffy
Friends
Zoro
Nami
Usopp


User Profile
127.0.0.1:5000/profile/2
User Profile
Name: Zoro
Friends
Luffy
Usopp
Nami

RESULT:

EX.NO: 02	CREATE A NETWORK MODEL USING NEO4J
DATE:	

AIM:

To create a network model using node4j.

ALGORITHM:

STEP 1: Start.

STEP 2: Download and install neo4j.

STEP 3: Open the Neo4j browser.

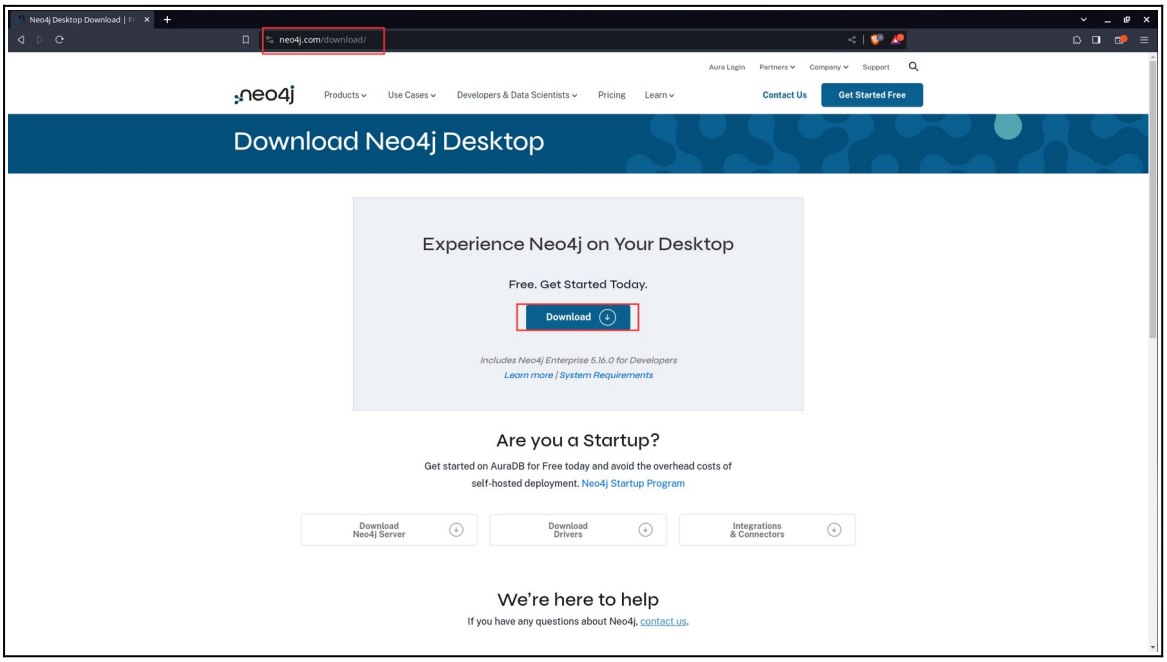
STEP 4: Create a new network model and retrieve the graph.

STEP 5: Stop.

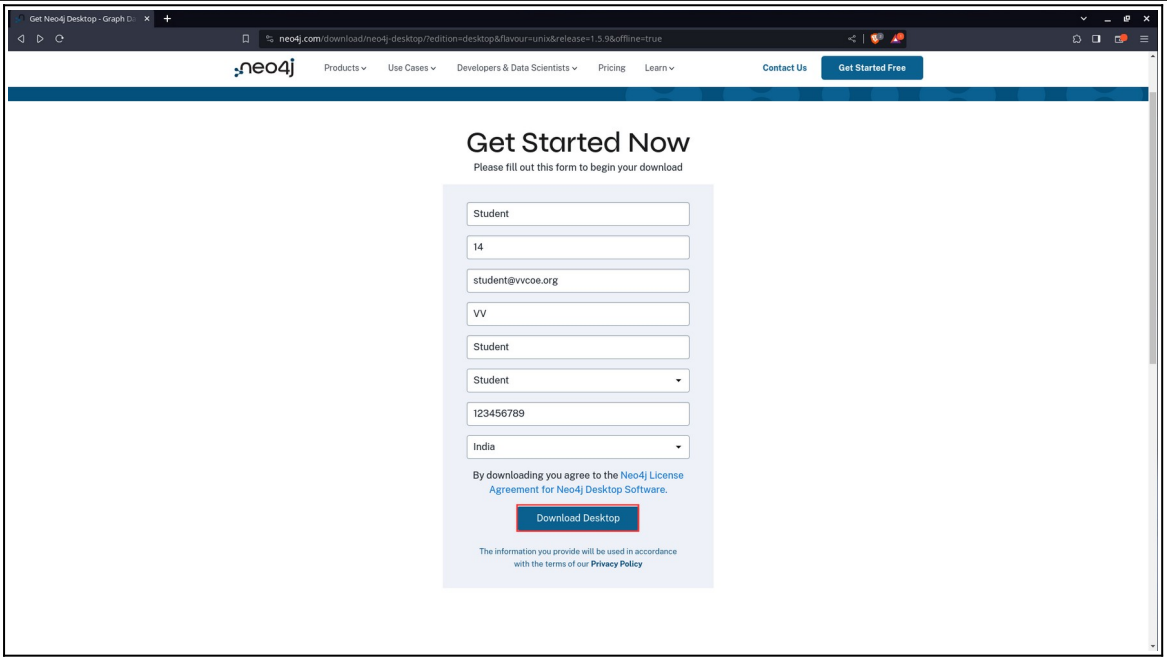
INSTALLATION:

Step 1: Navigate to the Neo4j download page by visiting https://neo4j.com/download/.

Step 2: On the page, locate and click on the 'Download' button."

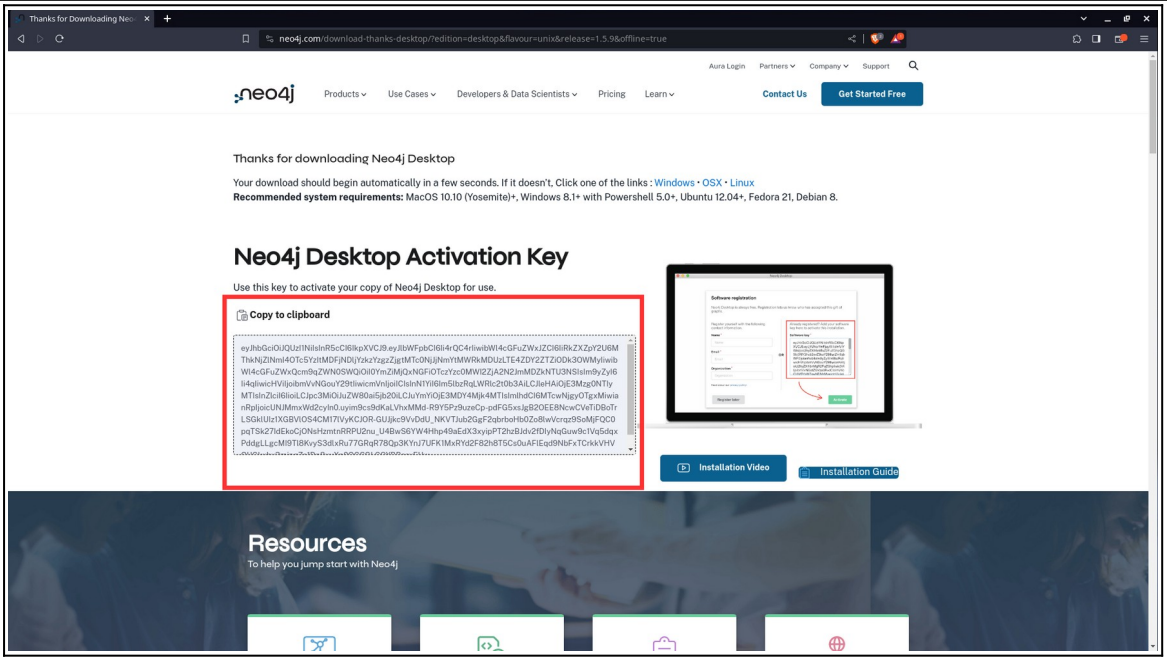


Step 3: Fill out the form and click “Download Desktop”.



(Note: The website automatically detects the desktop using the user-agent, and the suitable AppImage will begin downloading. Do not close the tab!)

Step 4: Copy the “Activation key” to the clipboard and wait for the download to finish.



Step 5: To start Neo4j, verify the downloaded file in the `~/Downloads` directory.

$ ls -al | grep “neo4j”

Change the permissions to make it executable.

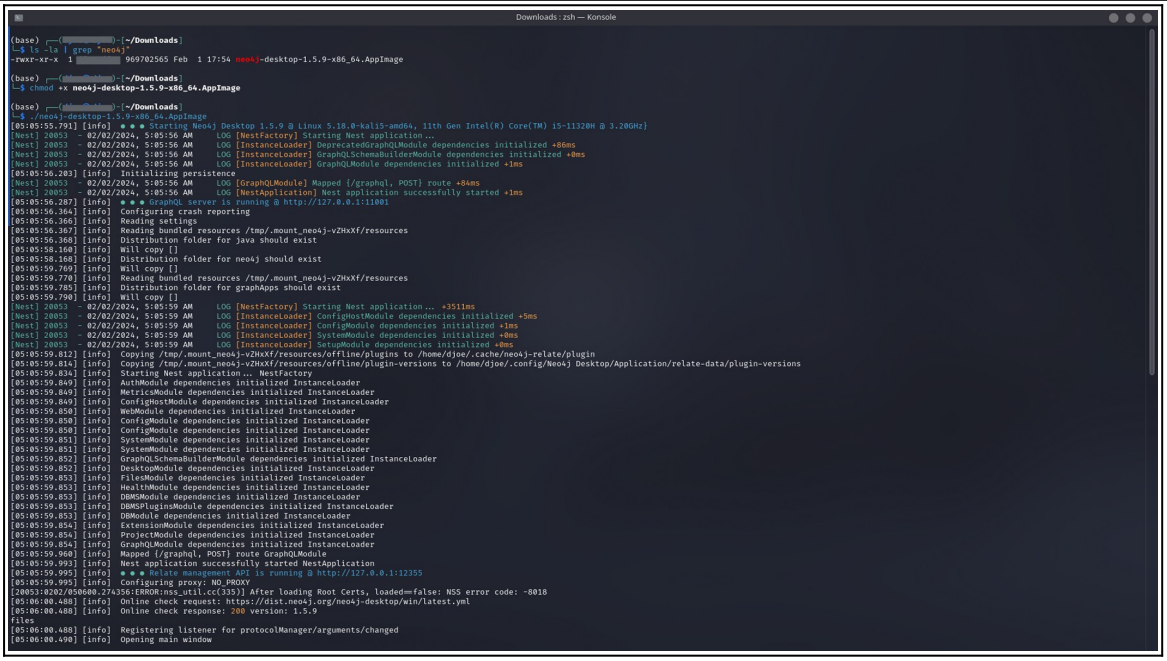
$ chmod +x neo4j-desktop-1.5.9-x86_64.AppImage

(Note: The “neo4j-desktop-1.5.9-x86_64.AppImage” may change according to the version you downloaded. Verify your AppImage name using “ls -al | grep “neo4j”)

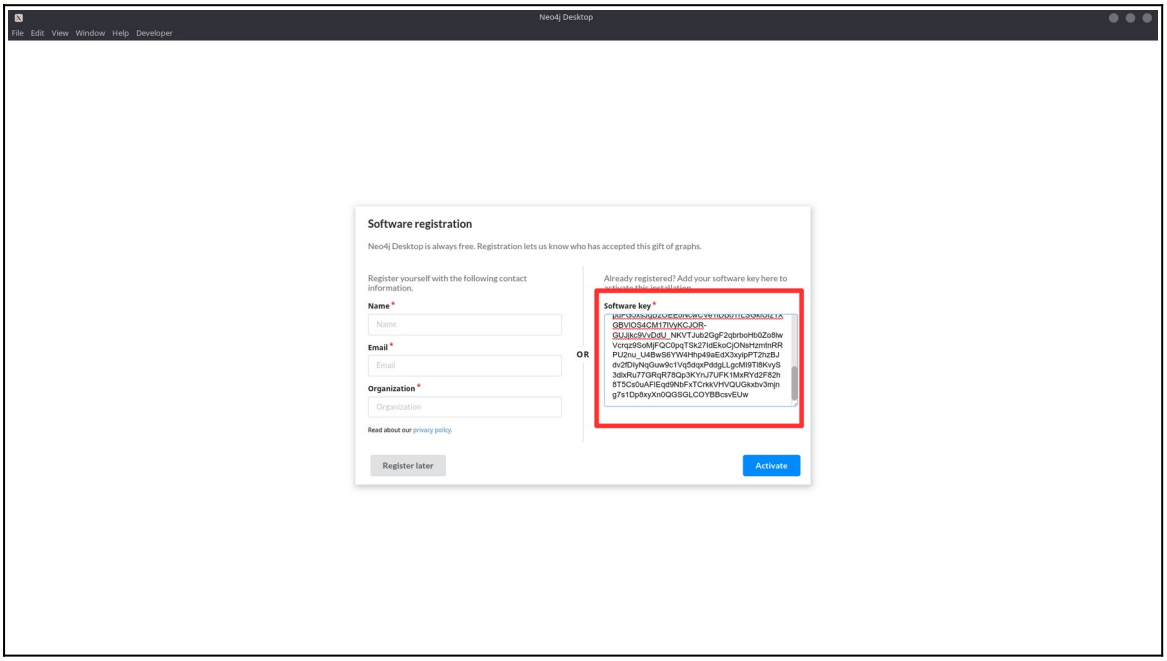
Start the AppImage:

$ ./neo4j-desktop-1.5.9-x86_64.AppImage

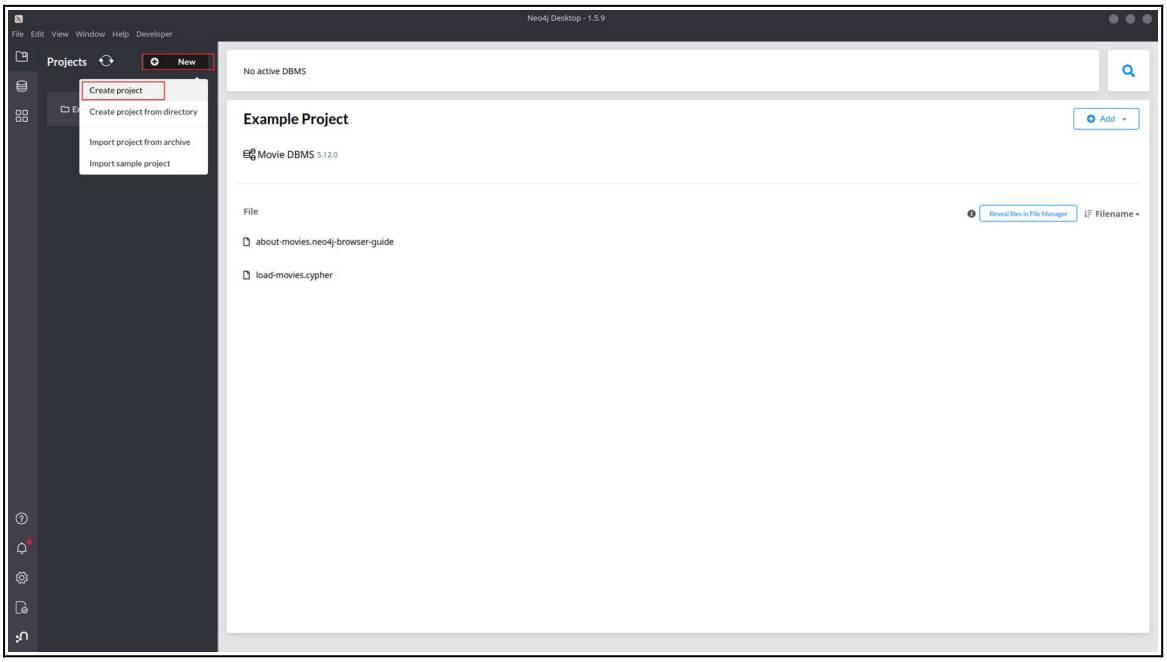
(Note: The “neo4j-desktop-1.5.9-x86_64.AppImage” may change according to the version you downloaded. Verify your AppImage name using “ls -al | grep “neo4j”)



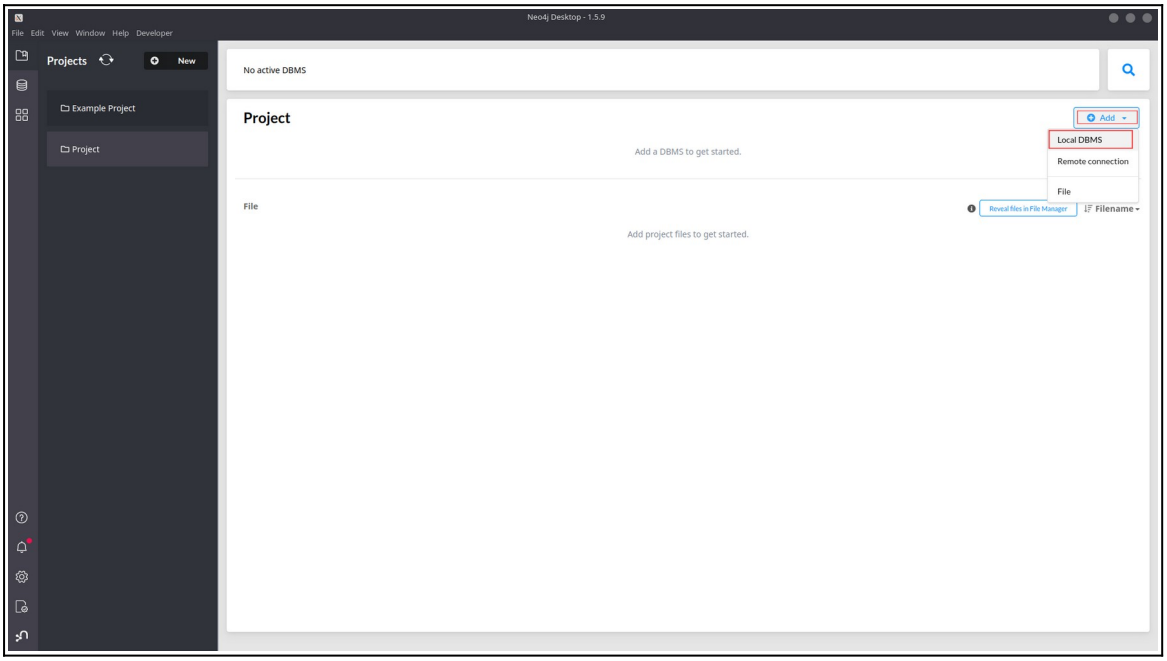
Step 6: After opening Neo4j, navigate to the 'Software Key' section and paste the previously copied 'Activation Key'. Then, click the 'Activate' button to complete the activation process.



Step 7: Within Neo4j, click on the 'New' button to create a new project.



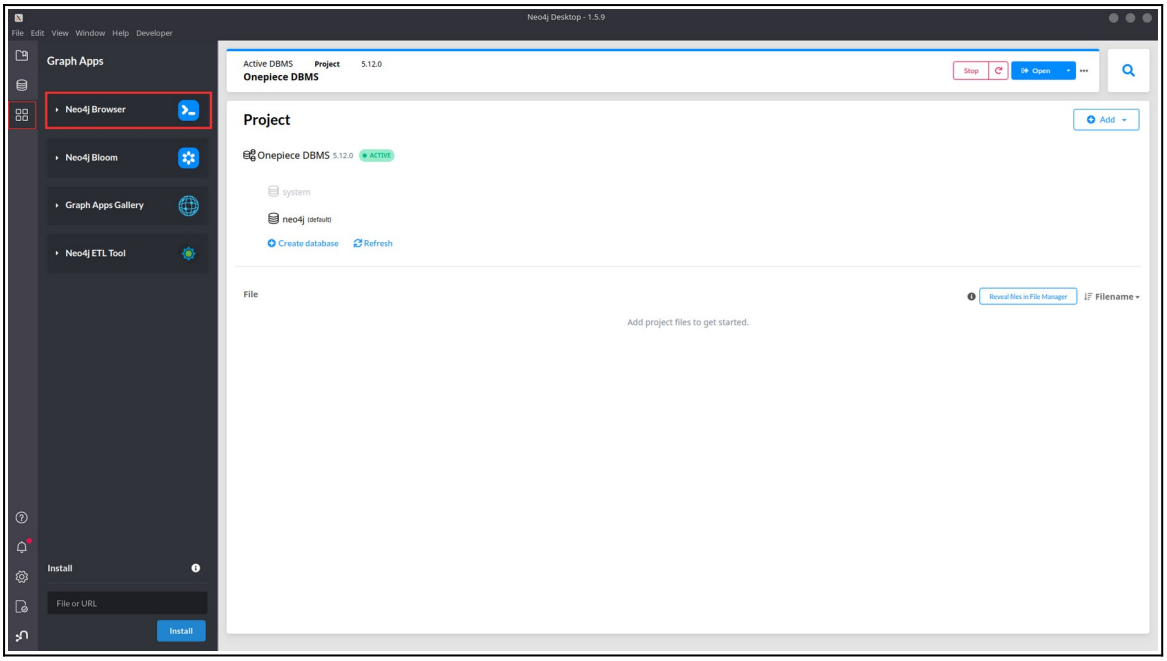
Step 8: Once you have created a new project, click on 'Add' and then select 'Local DBMS' to add a new database to our project.



When prompted, enter any desired DBMS name and password.

Step 9: Click the 'Start' button in the right corner of your newly created database.

Step 10: Once started, open the ‘Neo4j Browser’.



Once the Neo4j Browser has started successfully, this is where you can execute your 'Cypher query'.

PROGRAM:

Creating character nodes:

```
CREATE (:Character {name: 'Monkey D. Luffy', role: 'Main Protagonist'})
CREATE (:Character {name: 'Roronoa Zoro', role: 'Swordsman'})
CREATE (:Character {name: 'Nami', role: 'Navigator'})
CREATE (:Character {name: 'Usopp', role: 'Sniper'})
CREATE (:Character {name: 'Sanji', role: 'Cook'})
```

Creating crew relationship:

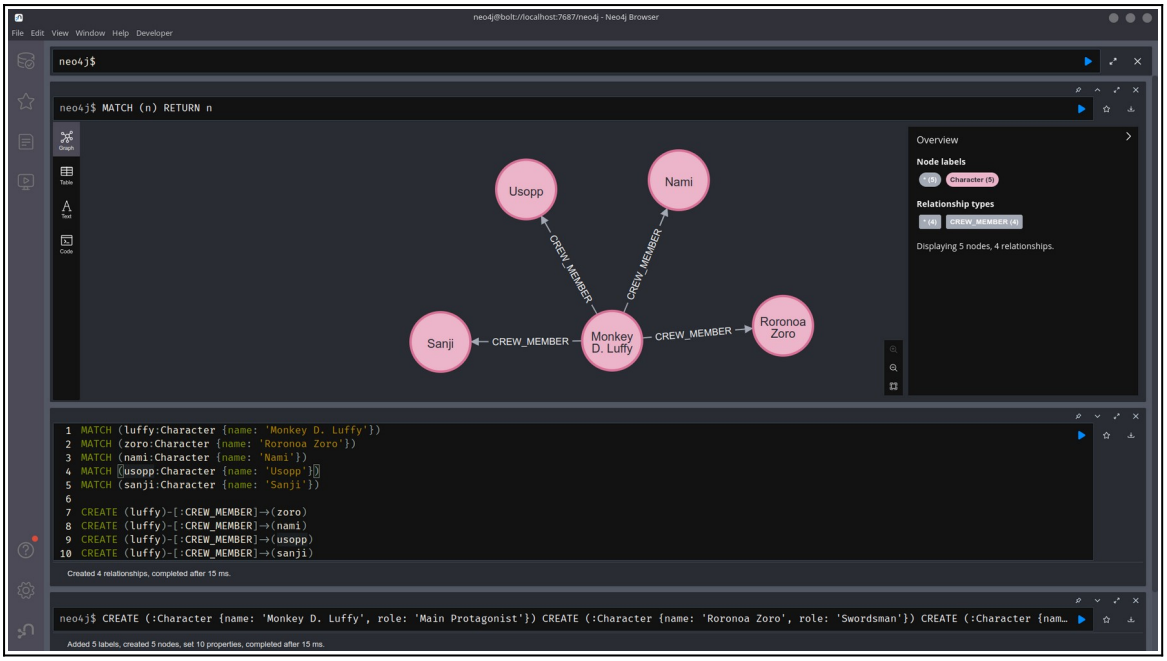
```
MATCH (luffy:Character {name: 'Monkey D. Luffy'})
MATCH (zoro:Character {name: 'Roronoa Zoro'})
MATCH (nami:Character {name: 'Nami'})
MATCH (usopp:Character {name: 'Usopp'})
MATCH (sanji:Character {name: 'Sanji'})

CREATE (luffy)-[:CREW_MEMBER]->(zoro)
CREATE (luffy)-[:CREW_MEMBER]->(nami)
CREATE (luffy)-[:CREW_MEMBER]->(usopp)
CREATE (luffy)-[:CREW_MEMBER]->(sanji)
```

Returning graph:

```
MATCH (n) RETURN n
```

(Interact with graph).

OUTPUT:



RESULT:

EX.NO: 03	READ AND WRITE DATA FROM GRAPH DATABASE
DATE:	

AIM:

To read and write data from graph database.

ALGORITHM:

STEP 1: Start.

STEP 2: Initiate the process by preparing for data management within the Neo4j graph database.

STEP 3: Utilize the `CREATE` command to seamlessly integrate new data into the graph database. This step involves the structured insertion of information, conforming to the predefined data model.

STEP 4: Employ the powerful `MATCH` clause to pinpoint specific data nodes or relationships within the graph. Further enhance the query by using the `RETURN` statement to elegantly present the desired information.

STEP 5: Stop.

PROGRAM:

Write data to graph database:

```
// nodes for characters
CREATE (:Character {name: 'Monkey D. Luffy', position: 'Captain'})
CREATE (:Character {name: 'Roronoa Zoro', position: 'Swordsman'})
CREATE (:Character {name: 'Nami', position: 'Navigator'})

// nodes for islands
CREATE (:Island {name: 'Dressrosa', type: 'Kingdom'})
CREATE (:Island {name: 'Alabasta', type: 'Kingdom'})
WITH 1 as dummy

// relationships between characters and islands
MATCH (luffy:Character {name: 'Monkey D. Luffy'})
MATCH (zoro:Character {name: 'Roronoa Zoro'})
MATCH (nami:Character {name: 'Nami'})
MATCH (dressrosa:Island {name: 'Dressrosa'})
MATCH (alabasta:Island {name: 'Alabasta'})

CREATE (luffy)-[:VISITS]->(dressrosa)
CREATE (zoro)-[:VISITS]->(alabasta)
CREATE (nami)-[:VISITS]->(dressrosa)
```

Reading data from graph database:

```
// Retrieve all characters
MATCH (c:Character)
RETURN c
```

Retrieve characters visiting a specific island:

```
MATCH (character)-[:VISITS]->(island:Island {name: 'Dressrosa'})
RETURN character
```

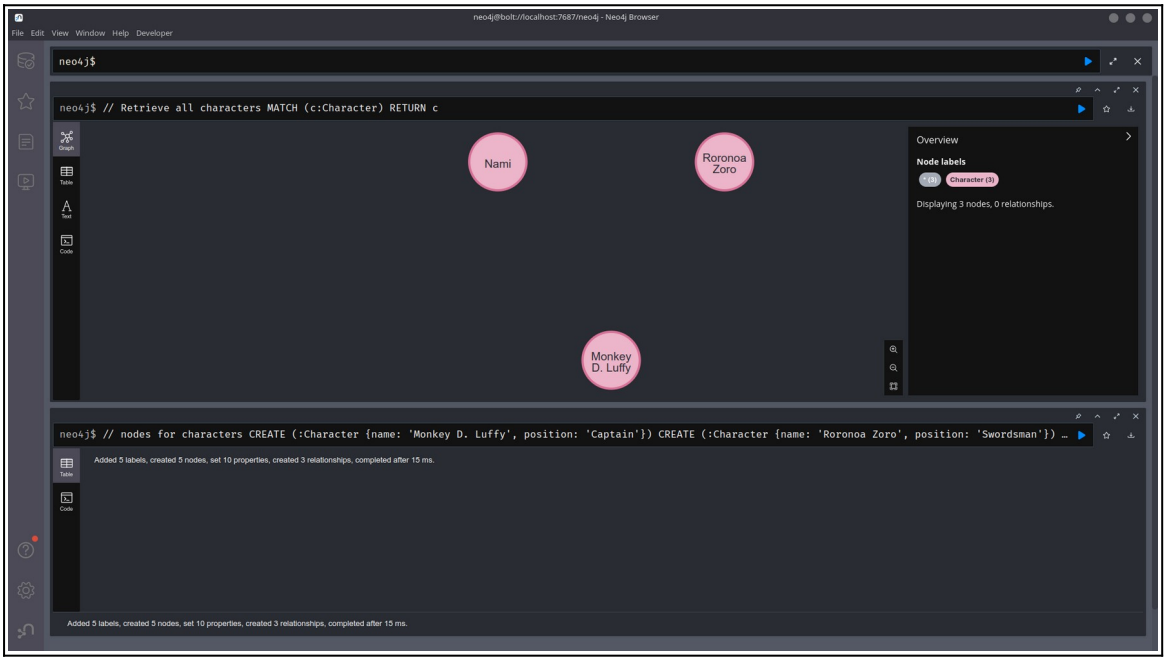
Updating data:

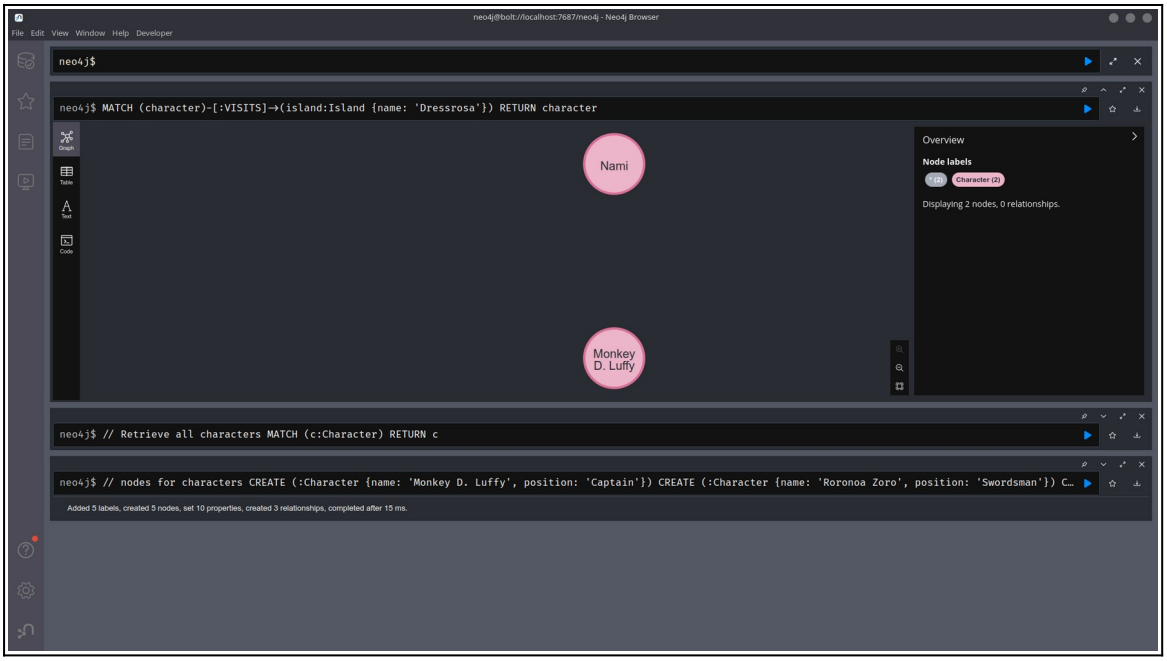
```
// Update character's position
MATCH (luffy:Character {name: 'Monkey D. Luffy'})
SET luffy.position = 'Pirate King'
RETURN luffy
```

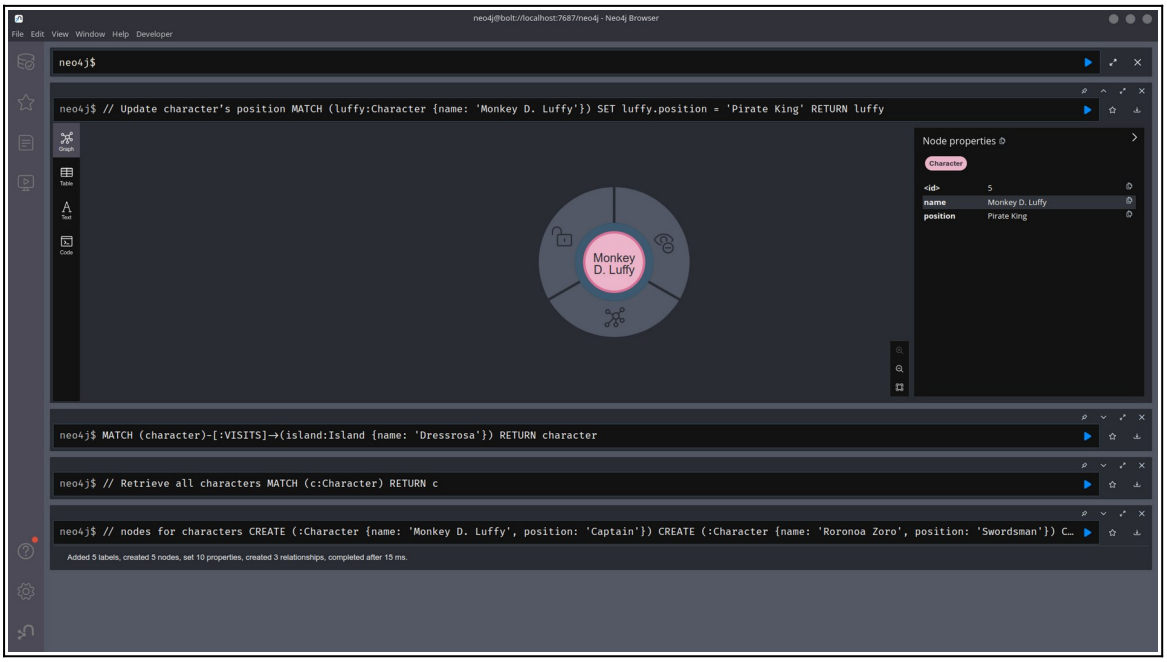
Deleting data:

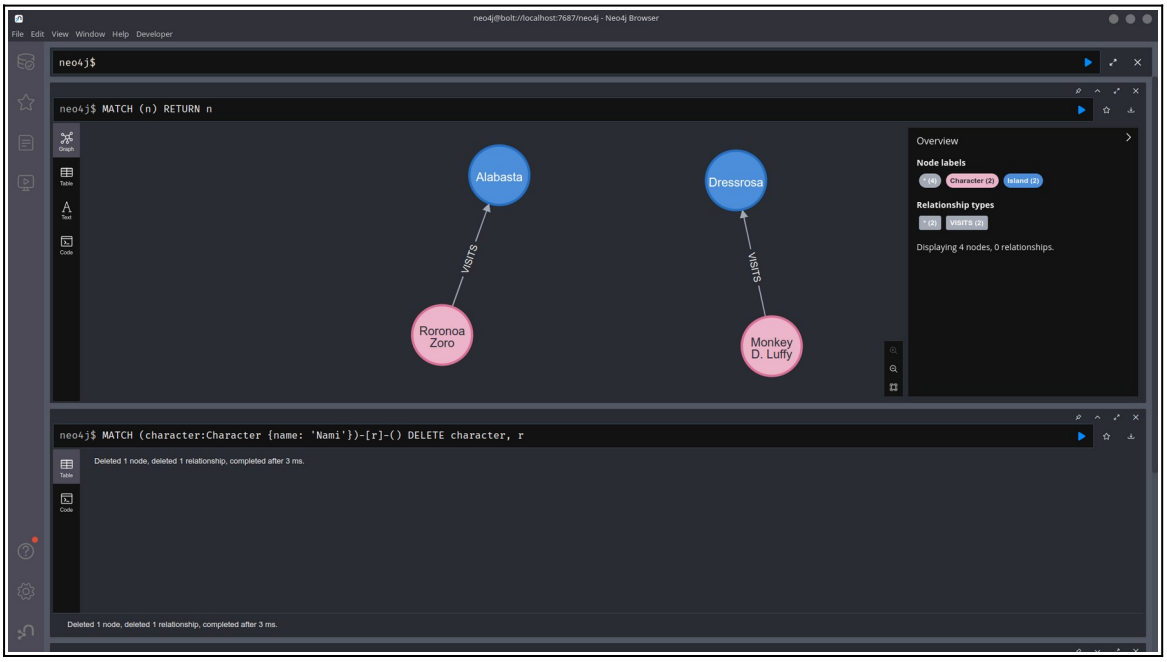
```
MATCH (character:Character {name: 'Nami'})-[r]-()
DELETE character, r
```

OUTPUT:


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
neo4j$ // Retrieve all characters MATCH (c:Character) RETURN c
Nami
Roronoa Zoro
Monkey D. Luffy
Overview
Node labels
Character (3)
Displaying 3 nodes, 0 relationships.
neo4j$ // nodes for characters CREATE (:Character {name: 'Monkey D. Luffy', position: 'Captain'}) CREATE (:Character {name: 'Roronoa Zoro', position: 'Swordsman'}) …
Added 5 labels, created 5 nodes, set 10 properties, created 3 relationships, completed after 15 ms.
Added 5 labels, created 5 nodes, set 10 properties, created 3 relationships, completed after 15 ms.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
neo4j$ MATCH (character)-[:VISITS]→(island:Island {name: 'Dressrosa'}) RETURN character
Nami
Monkey D. Luffy
Overview
Node labels
Character (2)
Displaying 2 nodes, 0 relationships.
neo4j$ // Retrieve all characters MATCH (c:Character) RETURN c
neo4j$ // nodes for characters CREATE (:Character {name: 'Monkey D. Luffy', position: 'Captain'}) CREATE (:Character {name: 'Roronoa Zoro', position: 'Swordsman'}) C…
Added 5 labels, created 5 nodes, set 10 properties, created 3 relationships, completed after 15 ms.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
neo4j$ // Update character's position MATCH (luffy:Character {name: 'Monkey D. Luffy'}) SET luffy.position = 'Pirate King' RETURN luffy
Monkey D. Luffy
Node properties
Character
<id> 5
name Monkey D. Luffy
position Pirate King
neo4j$ MATCH (character)-[:VISITS]→(island:Island {name: 'Dressrosa'}) RETURN character
neo4j$ // Retrieve all characters MATCH (c:Character) RETURN c
neo4j$ // nodes for characters CREATE (:Character {name: 'Monkey D. Luffy', position: 'Captain'}) CREATE (:Character {name: 'Roronoa Zoro', position: 'Swordsman'}) C…
Added 5 labels, created 5 nodes, set 10 properties, created 3 relationships, completed after 15 ms.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
neo4j$ MATCH (n) RETURN n
Alabasta
Dressrosa
VISITS
VISITS
Roronoa Zoro
Monkey D. Luffy
Overview
Node labels
Character (2)
Island (2)
Relationship types
VISITS (2)
Displaying 4 nodes, 0 relationships.
neo4j$ MATCH (character:Character {name: 'Nami'})-[r]-() DELETE character, r
Deleted 1 node, deleted 1 relationship, completed after 3 ms.
Deleted 1 node, deleted 1 relationship, completed after 3 ms.

RESULT:

EX.NO: 04	FIND “FRIEND OF FRIENDS” USING NEO4J
DATE:	

AIM:

To find “friend of friends” using neo4j.

ALGORITHM:

STEP 1: Start.

STEP 2: Initiate the process by preparing for data management within the Neo4j graph database.

STEP 3: Write and execute Cypher queries to create nodes for characters.

STEP 4: Write and execute Cypher queries to establish friendship relationships between characters.

STEP 5: Write and execute Cypher queries to find "Friend of Friends" for a specific character.

STEP 6: Write and execute Cypher queries to visualize the graph in Neo4j Browser.

STEP 7: Explore the graph.

STEP 8: Stop.

PROGRAM:

Create nodes for characters:

// Create nodes for characters

```
CREATE (luffy:Character {name: 'Monkey D. Luffy'})
CREATE (zoro:Character {name: 'Roronoa Zoro'})
CREATE (nami:Character {name: 'Nami'})
CREATE (usopp:Character {name: 'Usopp'})
CREATE (sanji:Character {name: 'Sanji'})
```

// Create relationship between characters

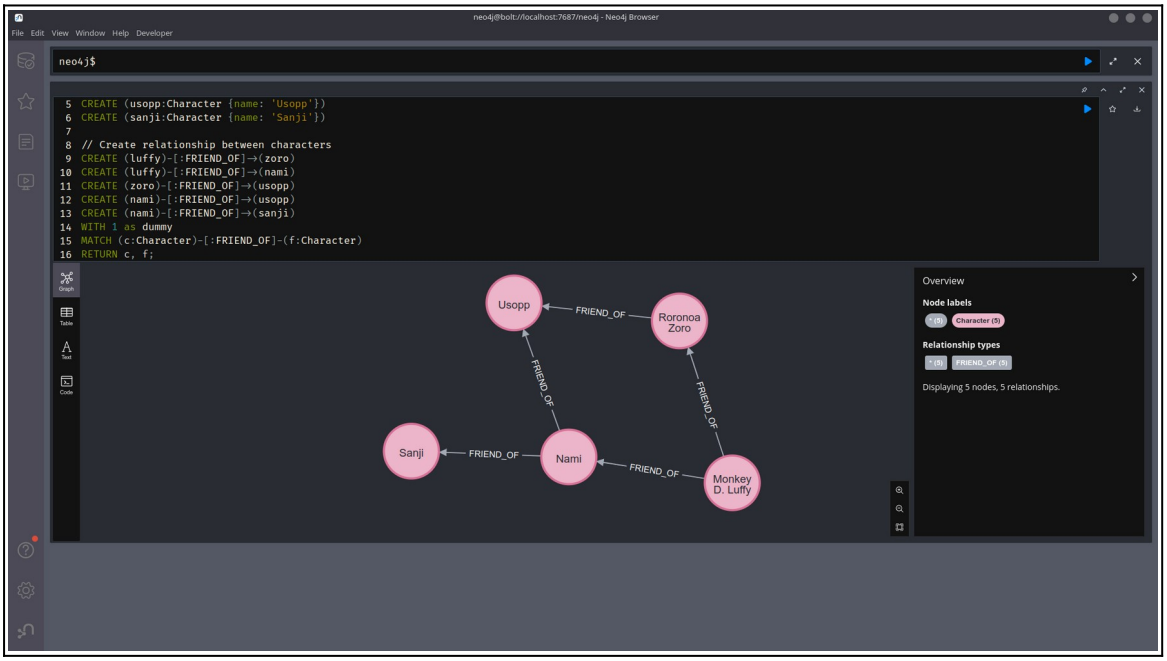
```
CREATE (luffy)-[:FRIEND_OF]→(zoro)
CREATE (luffy)-[:FRIEND_OF]→(nami)
CREATE (zoro)-[:FRIEND_OF]→(usopp)
CREATE (nami)-[:FRIEND_OF]→(usopp)
CREATE (nami)-[:FRIEND_OF]→(sanji)
WITH 1 as dummy
MATCH (c:Character)-[:FRIEND_OF]-(f:Character)
RETURN c, f;
```

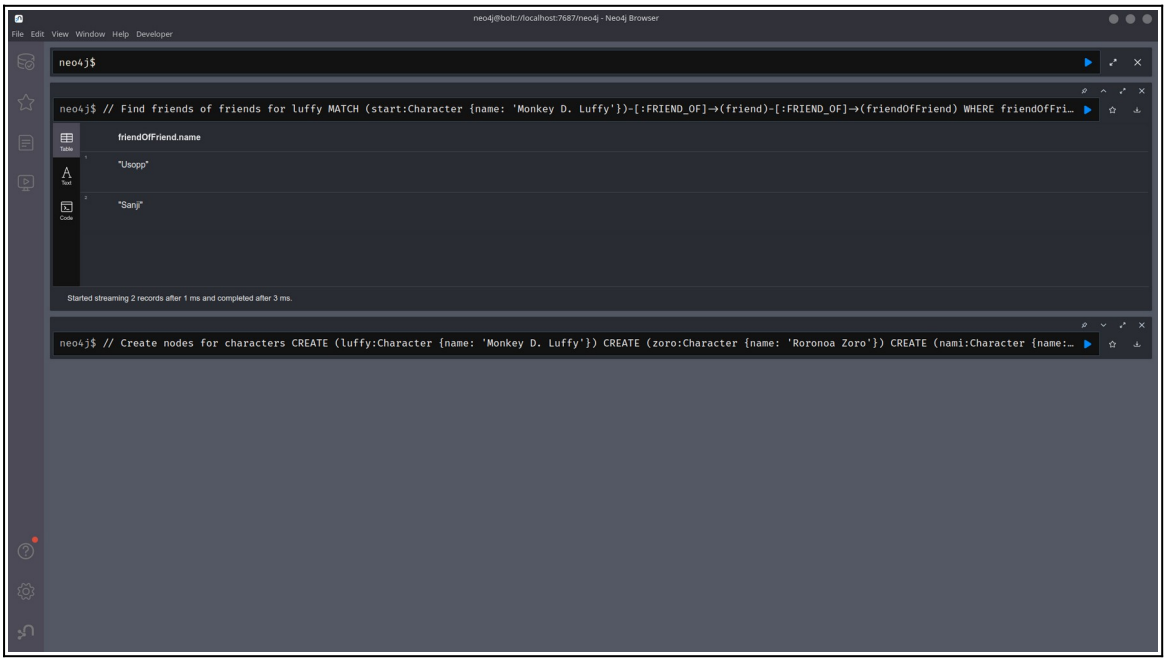
Finding friends of friends

// Find friends of friends for luffy

```
MATCH (start:Character {name: 'Monkey D. Luffy'})-[:FRIEND_OF]→(friend)-
[:FRIEND_OF]→(friendOfFriend)
WHERE friendOfFriend <> start
RETURN DISTINCT friendOfFriend.name
```

OUTPUT:


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
5 CREATE (usopp:Character {name: 'Usopp'})
6 CREATE (sanji:Character {name: 'Sanji'})
7
8 // Create relationship between characters
9 CREATE (luffy)-[:FRIEND_OF]→(zoro)
10 CREATE (luffy)-[:FRIEND_OF]→(nami)
11 CREATE (zoro)-[:FRIEND_OF]→(usopp)
12 CREATE (nami)-[:FRIEND_OF]→(usopp)
13 CREATE (nami)-[:FRIEND_OF]→(sanji)
14 WITH 1 as dummy
15 MATCH (c:Character)-[:FRIEND_OF]-(f:Character)
16 RETURN c, f;
Graph
Table
Text
Code
Usopp
FRIEND_OF
Roronoa Zoro
FRIEND_OF
FRIEND_OF
Sanji
FRIEND_OF
Nami
FRIEND_OF
Monkey D. Luffy
Overview
Node labels
* (5)
Character (5)
Relationship types
* (5)
FRIEND_OF (5)
Displaying 5 nodes, 5 relationships.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
neo4j$ // Find friends of friends for luffy MATCH (start:Character {name: 'Monkey D. Luffy'})-[:FRIEND_OF]→(friend)-[:FRIEND_OF]→(friendOfFriend) WHERE friendOfFri…
Table
Text
Code
friendOfFriend.name
1 "Usopp"
2 "Sanji"
Started streaming 2 records after 1 ms and completed after 3 ms.
neo4j$ // Create nodes for characters CREATE (luffy:Character {name: 'Monkey D. Luffy'}) CREATE (zoro:Character {name: 'Roronoa Zoro'}) CREATE (nami:Character {name:…

RESULT:

EX.NO: 05	IMPLEMENT SECURE SEARCH IN SOCIAL MEDIA
DATE:	

AIM:

To implement secure search in social media.

ALGORITHM:

STEP 1: Start.

STEP 2: Create a new project directory with subdirectories and files.

STEP 3: Import necessary modules in app.py.

STEP 4: Flask : the web framework used for building the web application.
render_template : A function from Flask that renders HTML templates. Re: Stands for Regular Expression used to match string patterns.

STEP 5: Define sample user data.

STEP 6: Define a search query sanitizer to securely get input from user.

STEP 7: Return the search results.

STEP 8: Stop.

PROGRAM:

app.py:

```
from flask import Flask, render_template, request, redirect, url_for
import re

app = Flask(__name__)

user_data = {
    "luffy": {"name": "Monkey D. Luffy", "role": "Captain", "goal": "King of pirates"},
    "zoro": {"name": "Roronoa Zoro", "role": "Swordsman", "goal": "World's greatest
swordsman"},
    "nami": {"name": "Nami", "role": "Navigator", "goal": "Map the entire world"},
    "usopp": {"name": "Usopp", "role": "Sniper", "goal": "Brave warrior of the sea"},
    "sanji": {"name": "Sanji", "role": "Chef", "goal": "Find the All Blue"},
    "chopper": {"name": "Tony Tony Chopper", "role": "Doctor", "goal": "Cure any
disease"},
    "robin": {"name": "Nico Robin", "role": "Archaeologist", "goal": "Learn the true
history"},
    "franky": {"name": "Franky", "role": "Shipwright", "goal": "Build the best ship"},
    "brook": {"name": "Brook", "role": "Musician", "goal": "Reunite with Laboon"},
    "jinbe": {"name": "Jinbe", "role": "Helmsman", "goal": "Achieve true justice"},
}

def query_sanitizer(query):
    sanitized_query = re.sub(r'[^\w\s]', '', query.strip()) or "query"
    return sanitized_query

@app.route("/")
def index():
    return render_template("index.html", users=user_data.items())

@app.route("/search", methods=["POST"])
def search():
    query = request.form.get("search_query")
    return redirect(url_for("search_results", query=query_sanitizer(query)))

@app.route("/search/<query>")
def search_results(query):
    results = [(key, user) for key, user in user_data.items() if query.lower() in
user['name'].lower()]
    return render_template("search_results.html", query=query, results=results)
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

templates/index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Secure Search</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <header>
        <form action="/search" method="post">
            <label for="search_query">Search:</label>
            <input type="text" name="search_query" id="search_query" placeholder="Type
here.." required>
            <button type="submit">Search</button>
        </form>
    </header>
    <div class="users-container">
        <h3>Your Friends:</h3>
        <br />
        <div class="users">
            <ul>
                {% for member, details in users %}
                    <li>{{ details.name }}</li>
                    <br />
                {% endfor %}
            </ul>
        </div>
    </div>
</body>
</html>
```

templates/search_results.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Search Results</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <h1>Search Results for "{{ query }}"</h1>
    <div class="results-container">
        <section>
            <ul>
                {% for member, details in results %}
                    <li>
                        <h3>{{ details.name }}</h3>
                        <ul>
                            <li><span>Role: </span>{{ details.role }}</li>
                            <li><span>Dream: </span>{{ details.goal }}</li>
                        </ul>
                    </li>
                {% endfor %}
            </ul>
        </section>
    </div>
</body>
</html>
```

static/styles.css:

```
* {
    padding: 0px;
    margin: 0px;
}

.form-body {
    display: flex;
    align-items: center;
    place-content: center;
    place-items: center;
    margin-top: 5%;
    flex-direction: column;
    padding: 30px;
}

.form-container {
    display: flex;
}
```

```
.form-container form {
    display: flex;
    flex-direction: column;
}

.form-container form input[type="text"], input[type="email"], input[type="password"] {
    width: 80%;
    border: none;
    border-bottom: 1px solid grey;
    height: 30px;
    outline: none;
}

.checkboxes {
    display: flex;
    flex-direction: row;
    margin-top: 10px;
    padding: 3px;
}

select {
    margin-bottom: 10px;
}

.form-body h2 {
    margin-bottom: 20px;
}

button {
    width: 90px;
}

.result-body {
    padding: 30px;
}
```

PROJECT STRUCTURE: (Just For Ref.)

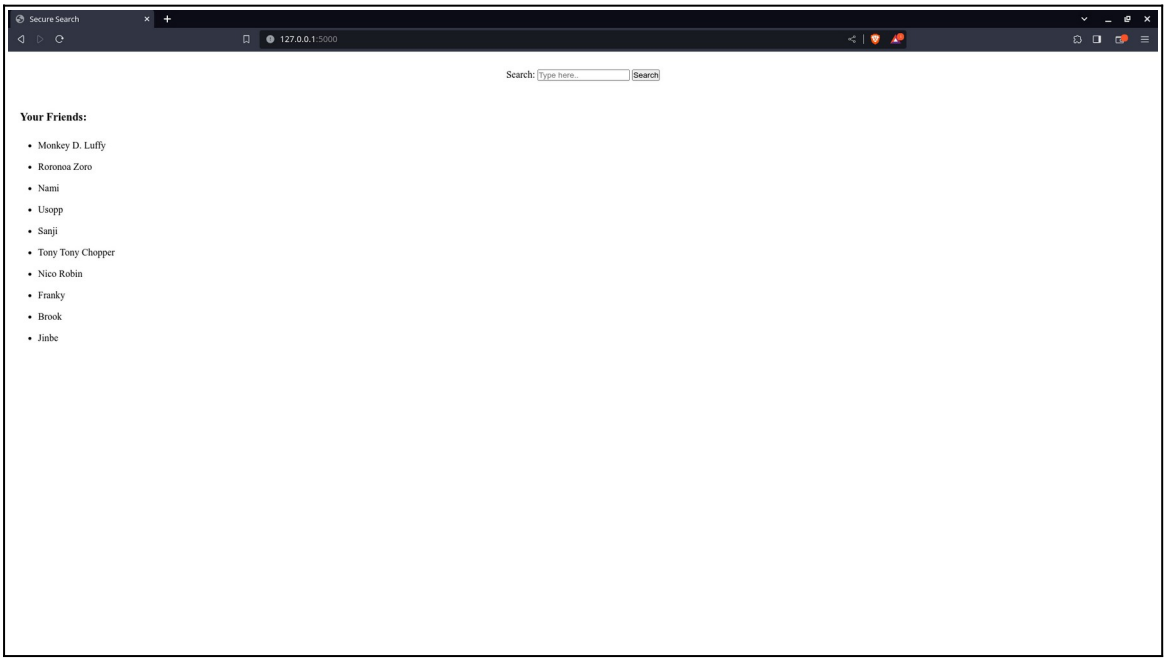
```
.
├── app.py
├── static
│   └── styles.css
└── templates
    ├── index.html
    └── search_results.html
```

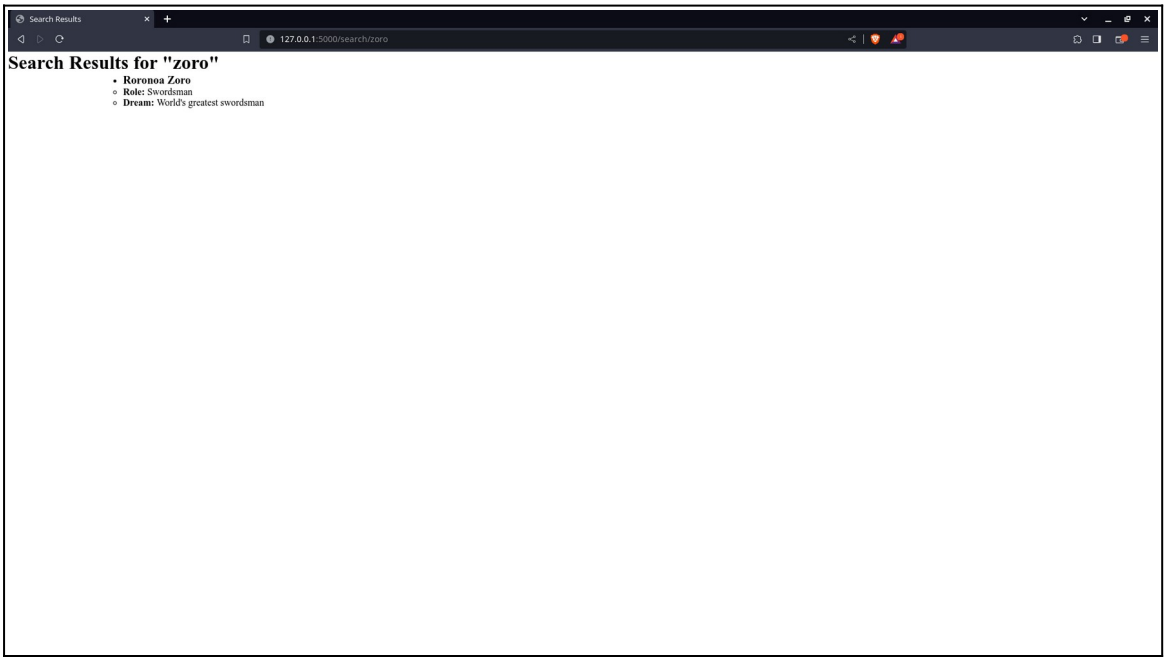
2 directories, 4 files

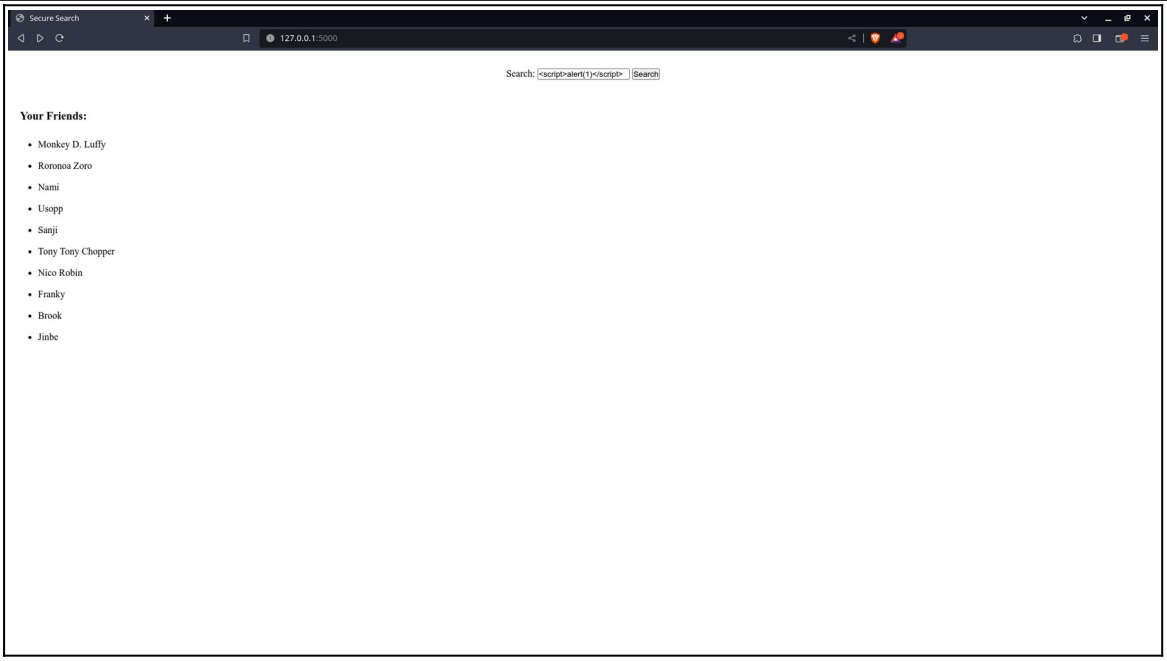
Execution (Python (Programming Language)):

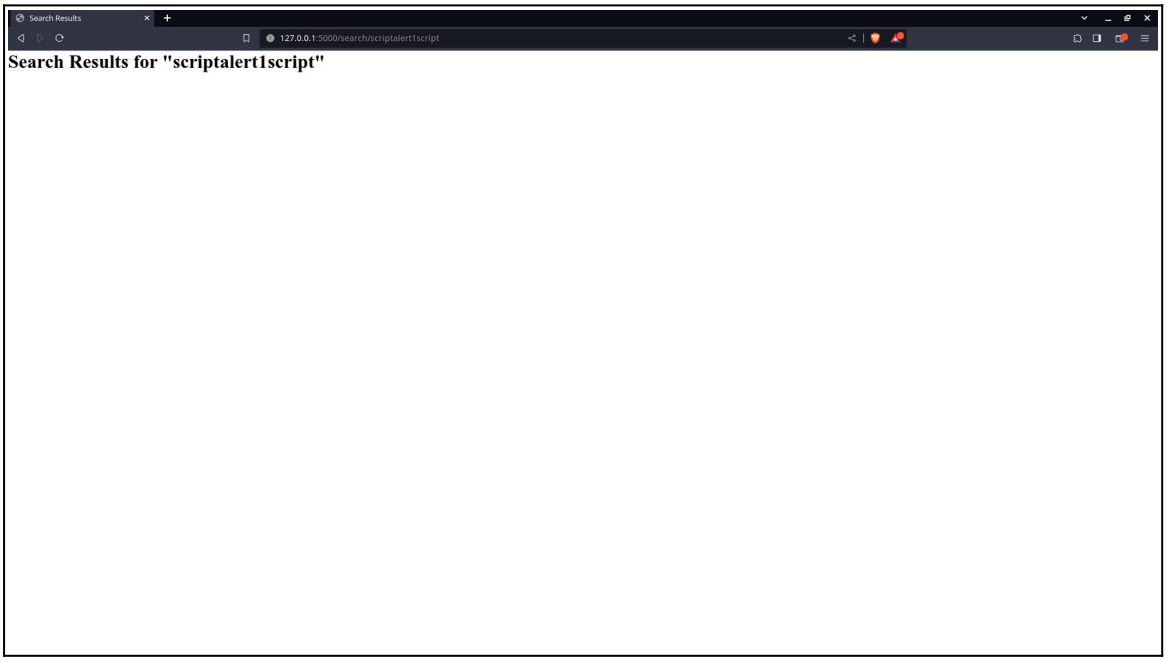
$ pip3 install Flask
$ python3 app.py

OUTPUT:


Secure Search
127.0.0.1:5000
Search: Type here.. Search
Your Friends:
Monkey D. Luffy
Roronoa Zoro
Nami
Usopp
Sanji
Tony Tony Chopper
Nico Robin
Franky
Brook
Jinbe


Search Results
127.0.0.1:5000/search/zoro
Search Results for "zoro"
Roronoa Zoro
Role: Swordsman
Dream: World's greatest swordsman


Secure Search
127.0.0.1:5000
Search: <script>alert(1)</script> Search
Your Friends:
Monkey D. Luffy
Roronoa Zoro
Nami
Usopp
Sanji
Tony Tony Chopper
Nico Robin
Franky
Brook
Jinbe


Search Results
127.0.0.1:5000/search/scriptalert1script
Search Results for "scriptalert1script"

RESULT:

EX.NO: 06	CREATE A SIMPLE SECURITY & PRIVACY DETECTOR
DATE:	

AIM:

To create a simple security and privacy detector.

ALGORITHM:

STEP 1: Start.

STEP 2: Create a new project directory with subdirectories and files.

STEP 3: Import necessary modules in app.py. Flask, re

STEP 4: Define index and result functions.

STEP 5: Get form data and calculate ratio in ‘result’ function.

STEP 6: Return the results.

STEP 7: Stop.

PROGRAM:

app.py:

```
from flask import Flask, render_template, request
import re

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/result", methods=["POST"])
def result():
    username = request.form.get("username")
    password = request.form.get("password")
    _2fa = request.form.get("2fa")
    private = request.form.get("private")
    fields_to_check = ["priv_activity", "priv_pfp", "priv_bio", "priv_call"]
    privacy_values = {field: request.form.get(field) for field in fields_to_check}

    security_level = sum([
        len(password) >= 8,
        bool(re.compile(r'[^a-zA-Z0-9\s]').search(password)),
        bool(re.compile(r'\d').search(password)),
        bool(_2fa),
    ])

    privacy_level = sum([
        bool(private),
        sum(2 if value == "nobody" else 1 for value in privacy_values.values() if value ==
"nobody"),
        sum(1 for value in privacy_values.values() if value == "friends"),
    ])

    sec_ratio = "{:.2f}/10".format((security_level / 4) * 10)
    priv_ratio = "{:.2f}/10".format((privacy_level / 9) * 10)

    data = {
        "username": username,
        "sec_ratio": sec_ratio,
        "priv_ratio": priv_ratio
    }
```

```
    return render_template("detected_result.html", data=data)

if __name__ == "__main__":
    app.run(debug=True)
```

templates/index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Security and Privacy Detector</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body class="form-body">
    <h2>Security and Privacy Detector</h2>

    <div class="form-container">
        <form action="/result" method="post">
            <input type="text" placeholder="Username" name="username" required>
            <input type="text" placeholder="First Name" name="first_name">
            <input type="text" placeholder="Last Name" name="last_name">
            <input type="email" placeholder="Email" name="email">

            <!-- Security -->
            <input type="password" placeholder="Password" name="password" required>

            <div class="checkboxes">
                <input type="checkbox" value="2fa" name="2fa" id="2fa">
                <label for="2fa">Two-Step Verification</label>
            </div>

            <!-- Privacy -->
            <div class="checkboxes">
                <input type="checkbox" value="private" id="private" name="private">
                <label for="private">Private Account</label>
            </div>

            <label for="priv_email">Who can see my email</label>
            <select name="priv_email" id="priv_email">
                <option value="everybody">Everybody</option>
                <option value="friends">My Friends</option>
```

```
            <option value="nobody">Nobody</option>
        </select>

        <label for="priv_activity">Last seen and online</label>
        <select name="priv_activity" id="priv_activity">
            <option value="everybody">Everybody</option>
            <option value="friends">My Friends</option>
            <option value="nobody">Nobody</option>
        </select>

        <label for="priv_pfp">Who can see my profile photo</label>
        <select name="priv_pfp" id="priv_pfp">
            <option value="everybody">Everybody</option>
            <option value="friends">My Friends</option>
            <option value="nobody">Nobody</option>
        </select>

        <label for="priv_bio">Who can see my bio</label>
        <select name="priv_bio" id="priv_bio">
            <option value="everybody">Everybody</option>
            <option value="friends">My Friends</option>
            <option value="nobody">Nobody</option>
        </select>

        <label for="priv_call">Who can call me</label>
        <select name="priv_call" id="priv_call">
            <option value="everybody">Everybody</option>
            <option value="friends">My Friends</option>
            <option value="nobody">Nobody</option>
        </select>

        <button type="submit">Detect</button>
    </form>
  </div>
</body>
</html>
```

templates/detected_result.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Security and Privacy Results</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body class="result-body">
    <h3>Hello, {{data.username}}...</h3>

    <br>

    <ul>
        <li><span style="font-weight: 700;">Your security level:</span>
{{data.sec_ratio}}</li>
        <li><span style="font-weight: 700;">Your privacy level :</span>
{{data.priv_ratio}}</li>
    </ul>
</body>
</html>
```

static/styles.css:

```
* {
    padding: 0px;
    margin: 0px;
}

.form-body {
    display: flex;
    align-items: center;
    place-content: center;
    place-items: center;
    margin-top: 5%;
    flex-direction: column;
    padding: 30px;
}

.form-container {
    display: flex;
}

.form-container form {
    display: flex;
    flex-direction: column;
}
```

```
.form-container form input[type="text"], input[type="email"], input[type="password"] {
    width: 80%;
    border: none;
    border-bottom: 1px solid grey;
    height: 30px;
    outline: none;
}

.checkboxes {
    display: flex;
    flex-direction: row;
    margin-top: 10px;
    padding: 3px;
}

select {
    margin-bottom: 10px;
}

.form-body h2 {
    margin-bottom: 20px;
}

button {
    width: 90px;
}

.result-body {
    padding: 30px;
}
```

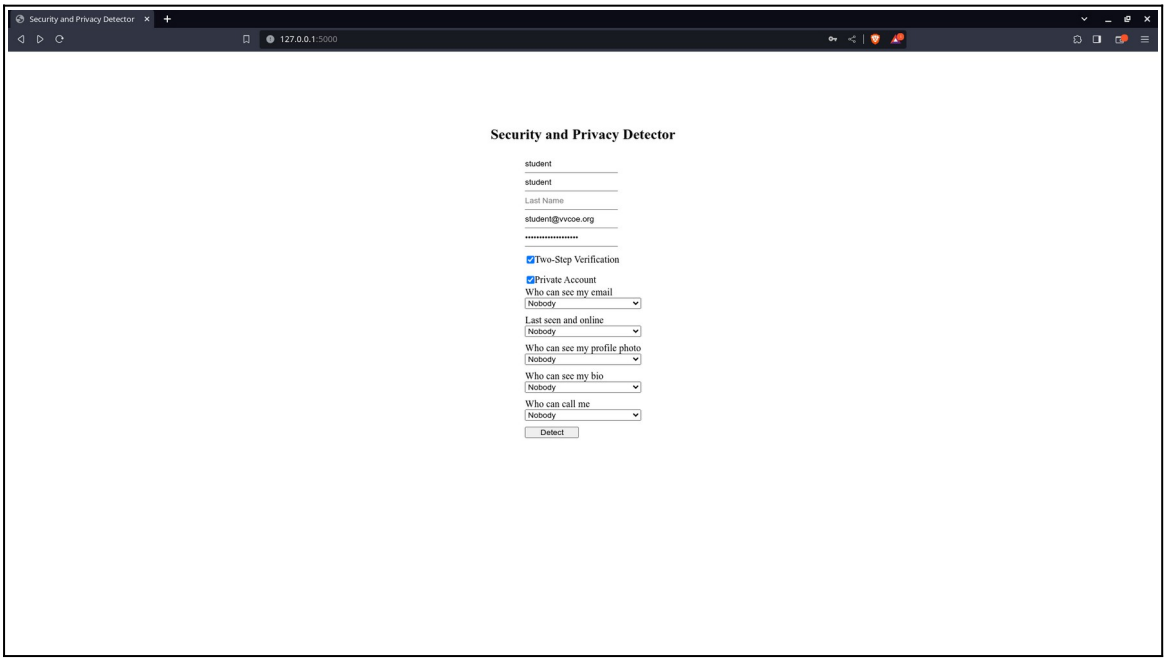
PROJECT STRUCTURE: (Just For Ref.)

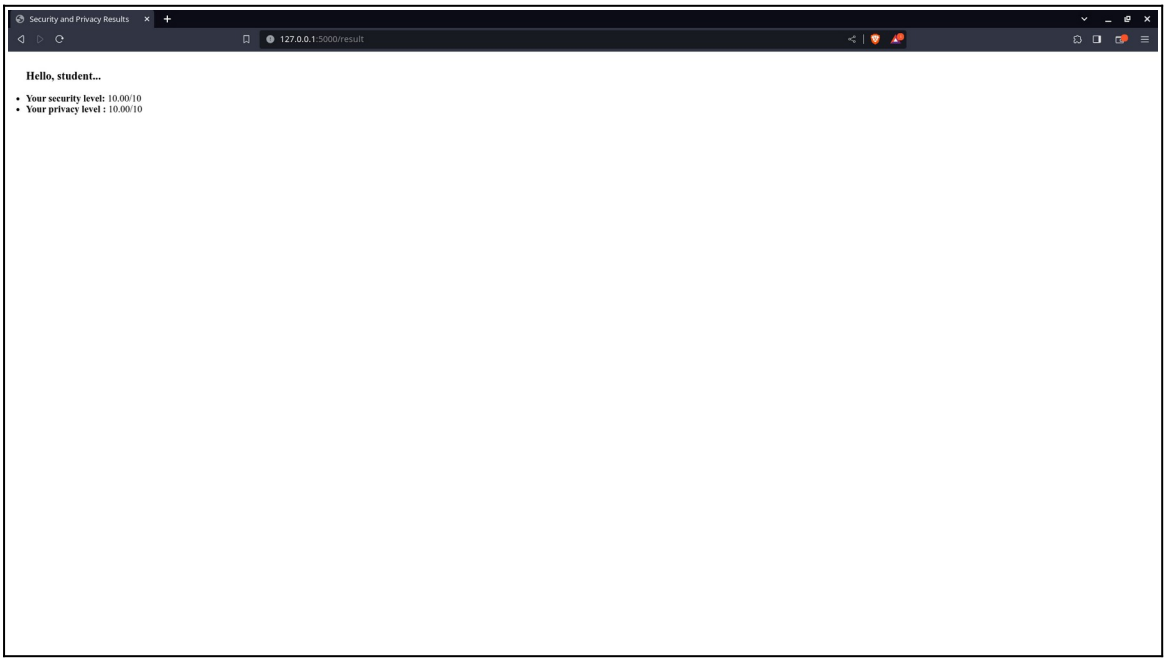
```
.
├── app.py
├── static
│   └── styles.css
└── templates
    ├── detected_result.html
    └── index.html

2 directories, 4 files
```

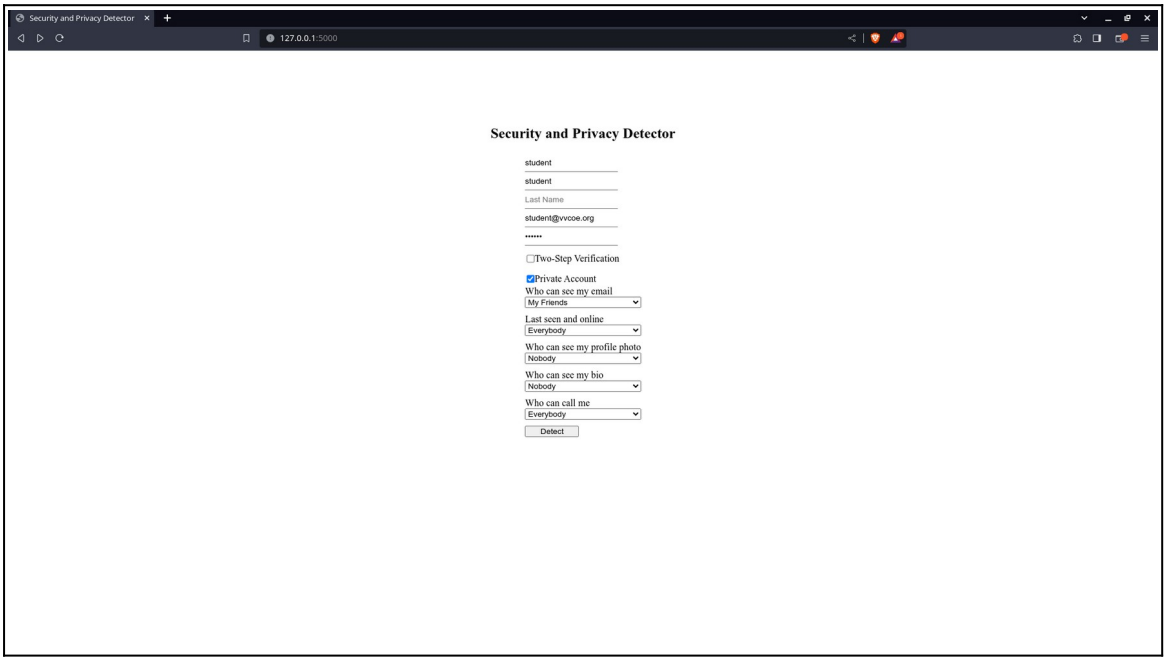
Execution (Python (Programming Language)):

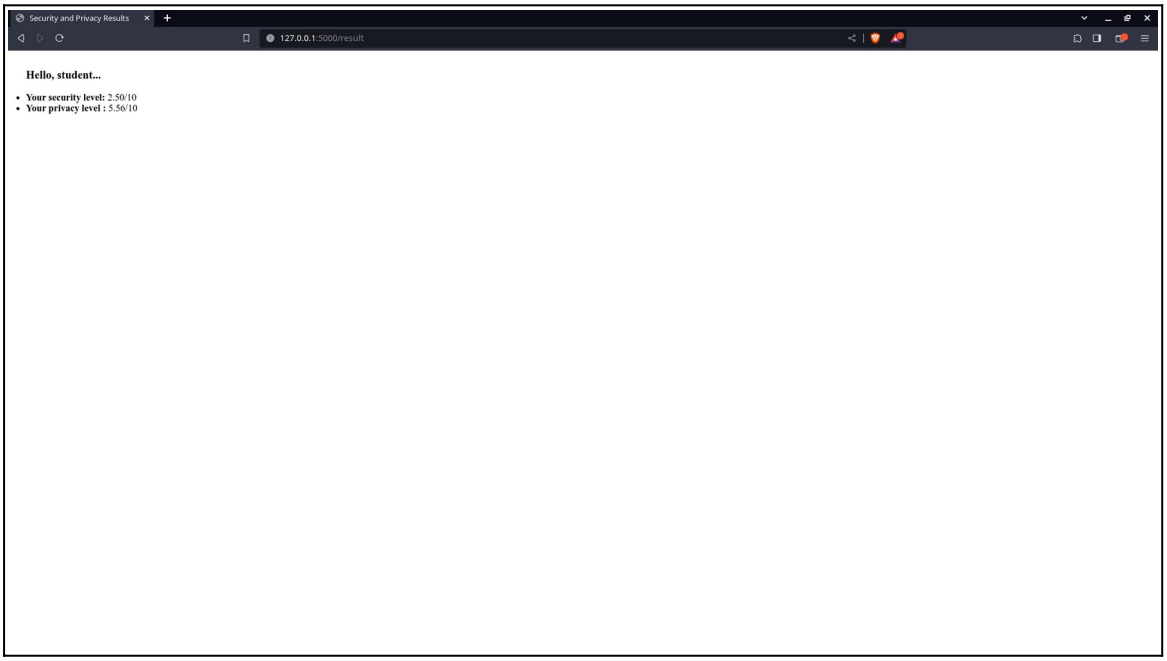
```
$ pip3 install Flask
$ python3 app.py
```

OUTPUT:


Security and Privacy Detector
127.0.0.1:5000
Security and Privacy Detector
student
student
Last Name
student@vvcoe.org
Two-Step Verification
Private Account
Who can see my email
Nobody
Last seen and online
Nobody
Who can see my profile photo
Nobody
Who can see my bio
Nobody
Who can call me
Nobody
Detect


Security and Privacy Results
127.0.0.1:5000/result
Hello, student...
Your security level: 10.00/10
Your privacy level : 10.00/10


Security and Privacy Detector
127.0.0.1:5000
Security and Privacy Detector
student
student
Last Name
student@vvcoe.org
Two-Step Verification
Private Account
Who can see my email
My Friends
Last seen and online
Everybody
Who can see my profile photo
Nobody
Who can see my bio
Nobody
Who can call me
Everybody
Detect


Security and Privacy Results
127.0.0.1:5000/result
Hello, student...
Your security level: 2.50/10
Your privacy level : 5.56/10

RESULT: