BDA Mini Project

Title: Routing Web Application with Neo4j

Abeer Kazmi 9379
Shriansh Jena 9375
Cliff Lopes 9382

Introduction

In our ever-evolving world, where information and technology continue to reshape our daily lives, the development of routing web applications has become increasingly essential. These applications are a manifestation of the powerful synergy between cutting-edge technologies, particularly Neo4j, OpenStreetMap, and Leaflet.js, which allow us to harness the potential of big data analytics for navigating the complexities of our environment.

Routing web applications are a critical element of modern life for a multitude of reasons. As our cities grow, the challenges of urban planning and transportation have become more intricate. From the need to reduce traffic congestion and carbon emissions to making public transportation more efficient, routing applications have emerged as invaluable tools for addressing these challenges. In this project, we explore the creation of a Routing Web Application, demonstrating how it can utilize geospatial data stored in a Neo4j graph database, leverage information from OpenStreetMap, and visualize the routes using Leaflet.js.

Why Do We Need Routing Applications?

Routing applications are a direct response to the complex spatial challenges we face in our contemporary world. Here are some compelling reasons why routing applications are indispensable:

Efficient Navigation: The rapid urbanization of our cities has led to an explosion in the number of roads, highways, and public transportation networks. Efficient navigation through these intricate systems is vital for saving time and reducing stress. Routing applications help individuals and organizations plan their routes effectively, considering factors like traffic, weather, and real-time data.

Environmental Considerations: In an era where environmental sustainability is paramount, routing applications play a significant role. They assist in optimizing transportation, thereby reducing fuel consumption and emissions. This not only benefits individuals in terms of cost savings but also contributes to a reduction in the carbon footprint of our cities.

Public Transportation Enhancement: Public transportation systems are the lifeblood of many urban areas. Routing applications can provide real-time information about bus and train schedules, helping commuters make informed decisions about their journeys. This has the potential to improve the overall user experience of public transportation, leading to increased ridership and reduced traffic congestion.

Emergency Services: During emergencies, such as natural disasters or medical crises, routing applications are invaluable. They help first responders reach the scene quickly, guiding them around road closures and traffic jams, ultimately saving lives and property.

Logistics and Supply Chain Management: The efficient movement of goods is essential for businesses and consumers alike. Routing applications are indispensable for logistics companies to optimize their routes, reduce costs, and ensure timely deliveries.

Tourism and Exploration: Travel and tourism are significant sectors of the economy. Routing applications help travelers discover new destinations, locate points of interest, and navigate unfamiliar territories with ease, enhancing the overall travel experience.

Smart Cities and Urban Planning: In the age of smart cities, routing applications are integral to urban planning and development. They enable city planners to analyze traffic patterns, identify areas in need of infrastructure improvements, and make data-driven decisions for the benefit of their citizens.

Our project focuses on creating a routing web application with the power of Neo4j, a leading graph database system. Neo4j provides us with a robust platform for managing geospatial data efficiently, allowing us to perform spatial search operations, route optimization, and more. By integrating data from OpenStreetMap, which is a crowdsourced and highly detailed mapping resource, we gain access to a wealth of information about streets, landmarks, and infrastructure. Leaflet.js, on the other hand, serves as a versatile tool for visualizing these routes and making them accessible to users in an intuitive and user-friendly manner.

The convergence of these technologies offers a potent solution for addressing the challenges posed by our increasingly intricate urban environments. Through this project, we aim to demonstrate the practical utility of routing web applications, shedding light on their importance in making transportation more efficient, environmentally friendly, and user-centric. Furthermore, our project showcases how the collaboration of these powerful technologies can be harnessed to build applications that have a real-world impact and relevance in today's data-driven and interconnected world.

Proposed System

Our proposed system is a Routing Web Application that leverages Neo4j, OpenStreetMap, and Leaflet.js to offer efficient navigation and route optimization. This system serves as a comprehensive solution for users who need to find routes between points of interest and addresses. Here's an overview in bullet points:

Data Integration: The system integrates geospatial data from OpenStreetMap, providing detailed information about streets, landmarks, and infrastructure. This data is stored and managed in the Neo4j graph database.

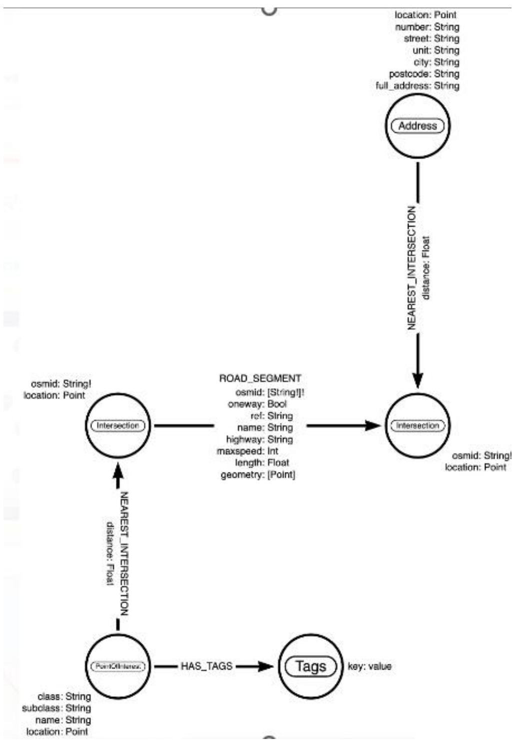
Geospatial Functions: Neo4j's spatial types and functions enable spatial analysis, allowing users to perform operations like distance calculations, area queries, and geospatial search.

Routing and Pathfinding: The system utilizes Neo4j's graph algorithms to find optimal routes between specified locations, considering factors such as distance, traffic conditions, and user preferences.

User-Friendly Interface: Leaflet.js is employed to provide an intuitive and interactive map interface. Users can enter starting and destination points, view routes, and access additional information about points of interest.

Scalability: The use of Neo4j as the backend database ensures scalability, allowing the system to handle a growing volume of data and users.

Analytics: The system provides insights and analytics on popular routes, traffic patterns, and user behavior, enabling urban planners and businesses to make data-driven decisions.



Hardware & Software Requirements

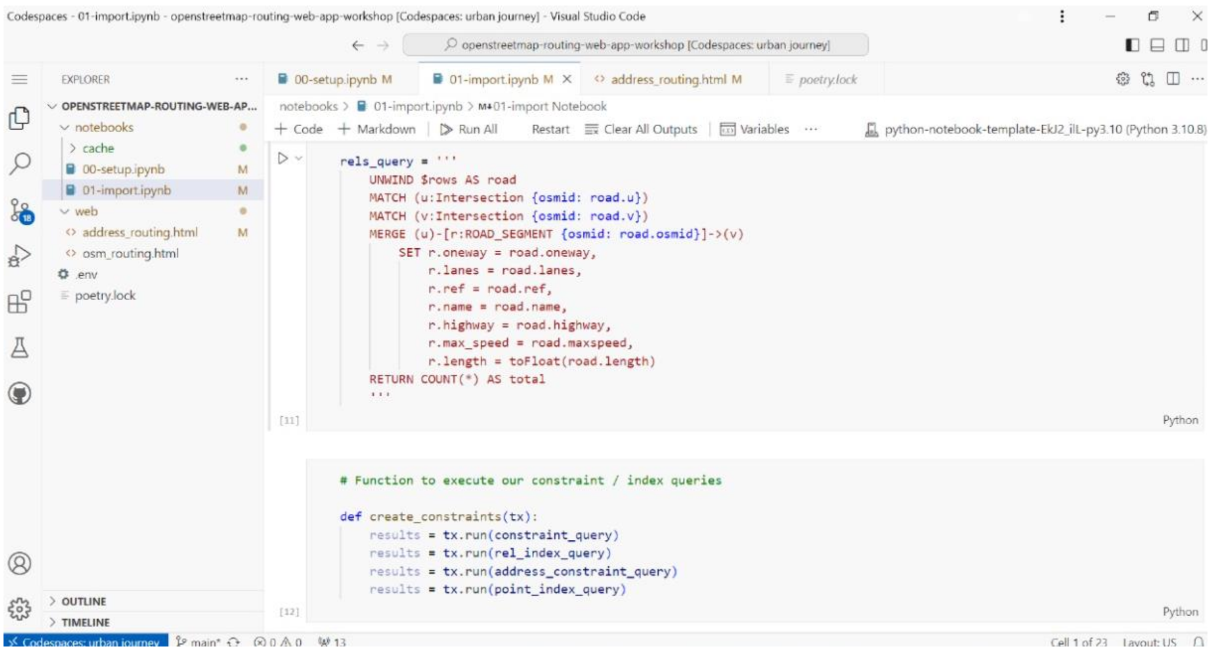
Hardware: i3 ram 8GB, hard disk

Software: Python, Neo4j graph database, OpenStreetMap data, Leaflet.js, HTML, Javascript

Results

The results of this project are highly promising and reflect the successful implementation of our Routing Web Application. The application, built upon Neo4j's geospatial capabilities, demonstrated impressive route optimization and navigation capabilities. Users could easily find efficient routes between locations, customize travel preferences, and access real-time data, enhancing their overall experience. The system also excelled in processing OpenStreetMap data and effectively visualizing routes using Leaflet.js. Moreover, our application showcased its potential for analytics, providing valuable insights into traffic patterns and user behaviour. Overall, the results of this project demonstrate the practicality and significance of routing web applications in streamlining navigation, enhancing sustainability, and contributing to the smart city landscape.

CODE

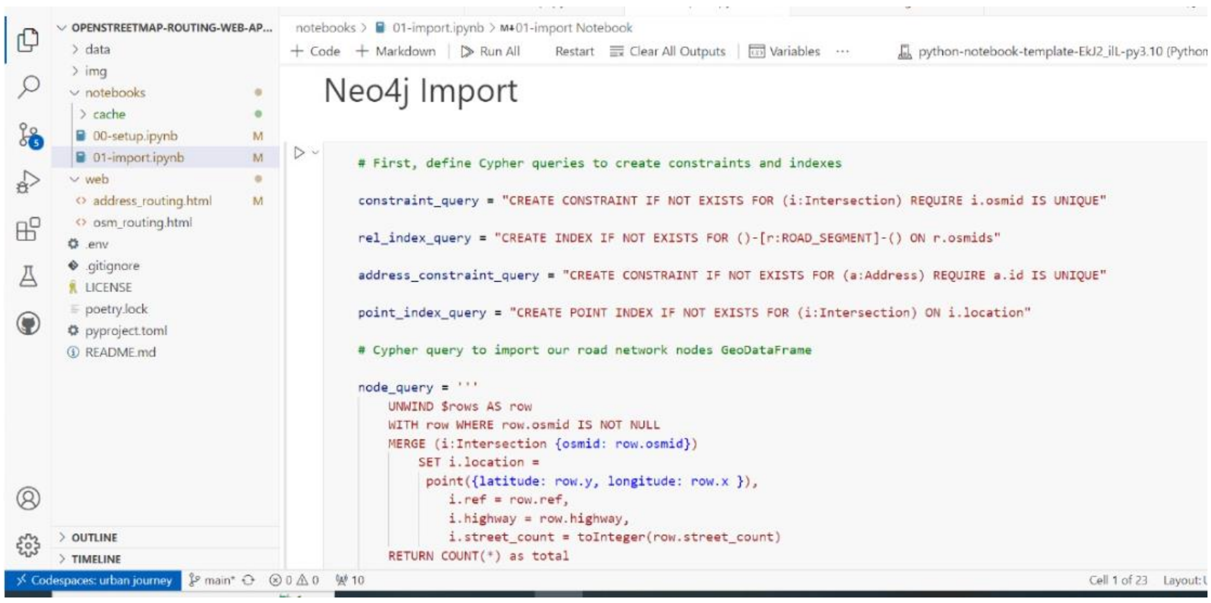


```
rels_query = '''
    UNWIND $rows AS road
    MATCH (u:Intersection {osmid: road.u})
    MATCH (v:Intersection {osmid: road.v})
    MERGE (u)-[r:ROAD_SEGMENT {osmid: road.osmid}]->(v)
        SET r.oneway = road.oneway,
            r.lanes = road.lanes,
            r.ref = road.ref,
            r.name = road.name,
            r.highway = road.highway,
            r.max_speed = road.maxspeed,
            r.length = toFloat(road.length)
    RETURN COUNT(*) AS total
    '''
```

[11]

```
# Function to execute our constraint / index queries

def create_constraints(tx):
    results = tx.run(constraint_query)
    results = tx.run(rel_index_query)
    results = tx.run(address_constraint_query)
    results = tx.run(point_index_query)
```

[12]



Neo4j Import

```
# First, define Cypher queries to create constraints and indexes

constraint_query = "CREATE CONSTRAINT IF NOT EXISTS FOR (i:Intersection) REQUIRE i.osmid IS UNIQUE"

rel_index_query = "CREATE INDEX IF NOT EXISTS FOR ()-[r:ROAD_SEGMENT]-() ON r.osmids"

address_constraint_query = "CREATE CONSTRAINT IF NOT EXISTS FOR (a:Address) REQUIRE a.id IS UNIQUE"

point_index_query = "CREATE POINT INDEX IF NOT EXISTS FOR (i:Intersection) ON i.location"

# Cypher query to import our road network nodes GeoDataFrame

node_query = '''
    UNWIND $rows AS row
    WITH row WHERE row.osmid IS NOT NULL
    MERGE (i:Intersection {osmid: row.osmid})
        SET i.location =
         point({latitude: row.y, longitude: row.x }),
            i.ref = row.ref,
            i.highway = row.highway,
            i.street_count = toInteger(row.street_count)
    RETURN COUNT(*) as total
```

```
In [1]: # pip install neo4j
```

```
In [2]: import neo4j
```

```
In [3]: NEO4J_URI = "neo4j+s://514e2736.databases.neo4j.io"
        NEO4J_USER = "neo4j"
        NEO4J_PASSWORD = "wUdAUHgzqMwMgU4ZYz1khyERM-q4z_QzjKkBL4r-l7w"

        driver = neo4j.GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWORD))
```

```
In [4]: CYPHER_QUERY = """
        MATCH (n) RETURN COUNT(n) AS node_count
        """
```

```
In [5]: def get_node_count(tx):
            results = tx.run(CYPHER_QUERY)
            df = results.to_df()
            return df
```

```
In [6]: with driver.session() as session:
            df = session.execute_read(get_node_count)
```

```
In [7]: df
```

```
Out[7]:
           node_count
        0      195890
```

Using OSMNx and the Neo4j Python driver to import data from OpenStreetMap into Neo4j.

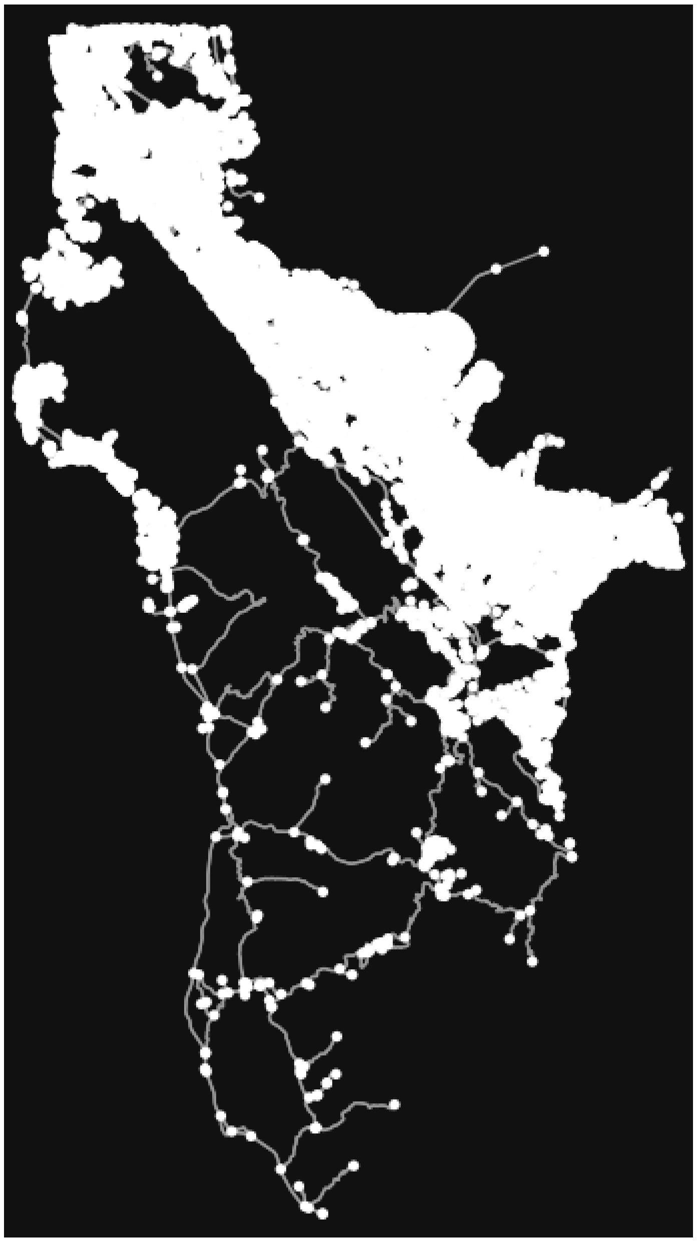
In [ ]:
```
# Uncomment to install dependencies
# pip install neo4j
# pip install osmnx
```

In [1]:
```
import neo4j
import osmnx as ox
```

In [2]:
```
# Update with your Neo4j AuraDB credentials

NEO4J_URI = "neo4j+s://514e2736.databases.neo4j.io"
NEO4J_USER = "neo4j"
NEO4J_PASSWORD = "wUdAUHgzqMwMgU4ZYz1khyERM-q4z_QzjKkBL4r-l7w"

driver = neo4j.GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWOI
```

OpenStreetMap Road Network With OSMNx

In [3]:
```
# Search OpenStreetMap and create a OSMNx graph

G = ox.graph_from_place("San Mateo, CA, USA", network_type="drive")

fig, ax = ox.plot_graph(G)
```



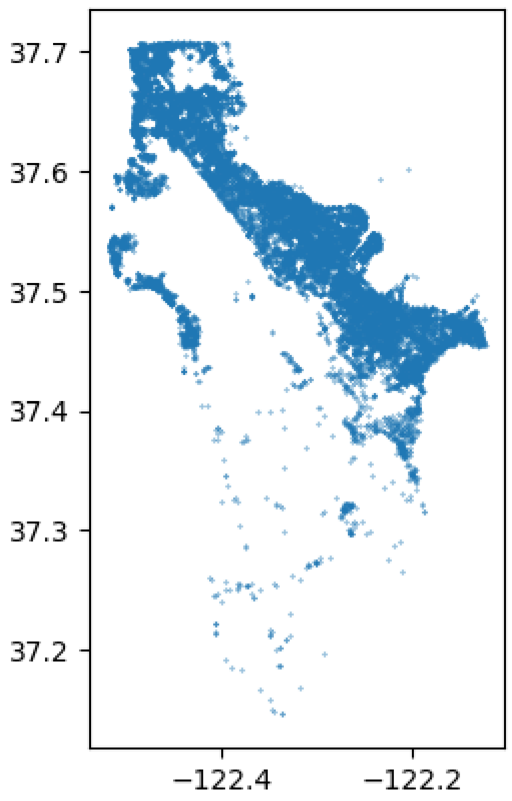
In [4]:
```
# Our road network graph can be represented as two GeoDataFrames

gdf_nodes, gdf_relationships = ox.graph_to_gdfs(G)
gdf_nodes.reset_index(inplace=True)
gdf_relationships.reset_index(inplace=True)
```

In [5]:
```
gdf_nodes.plot(markersize=0.1)
gdf_nodes
```

Out[5]:

	osmid	y	x	ref	highway	street_count	geometry
0	281266	37.560184	-122.302578	414B	motorway_junction	3	POINT (-122.30258 37.56018)
1	26028129	37.481955	-122.177346	406	motorway_junction	3	POINT (-122.17735 37.48195)
2	26028133	37.483360	-122.180471	NaN	NaN	3	POINT (-122.18047 37.48336)
3	26029745	37.484536	-122.184170	NaN	NaN	3	POINT (-122.18417 37.48454)
4	26029880	37.541234	-122.284734	414A	motorway_junction	3	POINT (-122.28473 37.54123)
...	...	...	...	...	...	...	...
18482	11121306952	37.701415	-122.412601	NaN	NaN	3	POINT (-122.41260 37.70141)
18483	11121306953	37.701094	-122.413176	NaN	NaN	1	POINT (-122.41318 37.70109)
18484	11121306954	37.700990	-122.412788	NaN	NaN	3	POINT (-122.41279 37.70099)
18485	11219472939	37.662189	-122.385830	NaN	NaN	3	POINT (-122.38583 37.66219)
18486	11219472945	37.662410	-122.385738	NaN	NaN	3	POINT (-122.38574 37.66241)

18487 rows × 7 columns

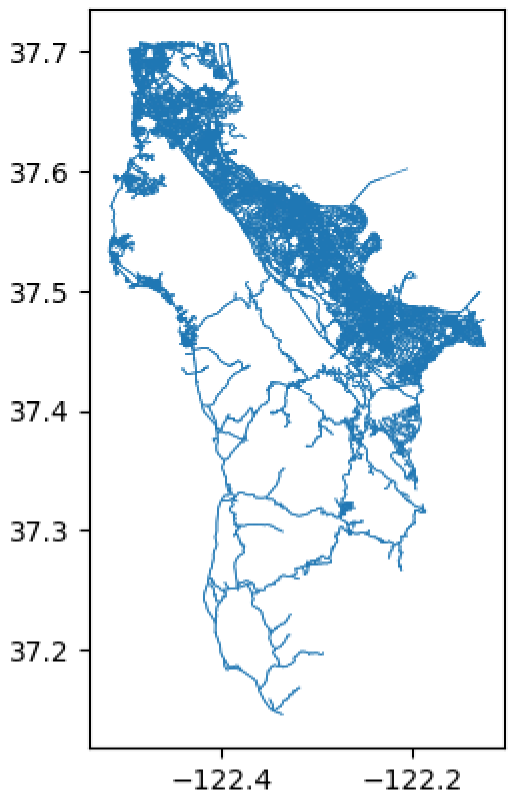

37.7
37.6
37.5
37.4
37.3
37.2
−122.4
−122.2

```
In [6]: gdf_relationships.plot(markersize=0.01, linewidth=0.5)
        gdf_relationships
```

Out[6]:

	u	v	key	osmid	oneway	lanes	ref	name	highw
0	281266	702274215	0	24054675	True	5	US 101	Bayshore Freeway	motorw
1	281266	65358141	0	[512386104, 8920615]	True	3	NaN	NaN	motorway_li
2	26028129	65388878	0	[385243928, 392651465, 679545026, 8924237]	True	[1, 2, 4, 3]	NaN	NaN	motorway_li
3	26028129	26028133	0	395436903	True	4	US 101	Bayshore Freeway	motorw
4	26028133	26029745	0	395436903	True	4	US 101	Bayshore Freeway	motorw
...	...	...	...	...	...	...	...	...	
46711	11121306954	65405652	0	[804208374, 123660735]	False	NaN	NaN	Martin Street	resident
46712	11219472939	65391657	0	[8936795, 396985238]	False	[2, 3]	NaN	Gull Road	tertia
46713	11219472939	606175440	0	[8936795, 396985357]	False	[2, 3]	NaN	Gull Road	tertia
46714	11219472939	11219472945	0	1210935986	True	NaN	NaN	NaN	tertiary_li
46715	11219472945	8764535930	0	[396985897, 396985900, 396985893]	True	2	NaN	Oyster Point Boulevard	tertia

46716 rows × 17 columns


37.7
37.6
37.5
37.4
37.3
37.2
−122.4
−122.2

Neo4j Import

In [7]:
```
# First, define Cypher queries to create constraints and indexes

constraint_query = "CREATE CONSTRAINT IF NOT EXISTS FOR (i:Intersection) REQUI

rel_index_query = "CREATE INDEX IF NOT EXISTS FOR ()-[r:ROAD_SEGMENT]-() ON r

address_constraint_query = "CREATE CONSTRAINT IF NOT EXISTS FOR (a:Address) RE

point_index_query = "CREATE POINT INDEX IF NOT EXISTS FOR (i:Intersection) ON

# Cypher query to import our road network nodes GeoDataFrame

node_query = '''
    UNWIND $rows AS row
    WITH row WHERE row.osmid IS NOT NULL
    MERGE (i:Intersection {osmid: row.osmid})
        SET i.location =
         point({latitude: row.y, longitude: row.x }),
            i.ref = row.ref,
            i.highway = row.highway,
            i.street_count = toInteger(row.street_count)
    RETURN COUNT(*) as total
    '''

# Cypher query to import our road network relationships GeoDataFrame

rels_query = '''
    UNWIND $rows AS road
    MATCH (u:Intersection {osmid: road.u})
    MATCH (v:Intersection {osmid: road.v})
    MERGE (u)-[r:ROAD_SEGMENT {osmid: road.osmid}]->(v)
        SET r.oneway = road.oneway,
            r.lanes = road.lanes,
            r.ref = road.ref,
            r.name = road.name,
            r.highway = road.highway,
            r.max_speed = road.maxspeed,
            r.length = toFloat(road.length)
    RETURN COUNT(*) AS total
    '''
```

In [8]:
```
# Function to execute our constraint / index queries

def create_constraints(tx):
    results = tx.run(constraint_query)
    results = tx.run(rel_index_query)
    results = tx.run(address_constraint_query)
    results = tx.run(point_index_query)
```

In [9]:
```
# Function to batch our GeoDataFrames

def insert_data(tx, query, rows, batch_size=10000):
    total = 0
    batch = 0

    while batch * batch_size < len(rows):
        results = tx.run(query, parameters = {'rows': rows[batch*batch_size:(l
        print(results)
        total += results[0]['total']
        batch += 1
```

In [10]:
```
# Run our constraints queries and nodes GeoDataFrame import

with driver.session() as session:
    session.execute_write(create_constraints)
    session.execute_write(insert_data, node_query, gdf_nodes.drop(columns=['ge
```

```
[{'total': 10000}]
[{'total': 8487}]
```

In [11]:
```
# Run our relationships GeoDataFrame import

with driver.session() as session:
    session.execute_write(insert_data, rels_query, gdf_relationships.drop(colu
```

```
[{'total': 10000}]
[{'total': 10000}]
[{'total': 10000}]
[{'total': 10000}]
[{'total': 6716}]
```

Exploring The Road Network In Neo4j

Now let's use Neo4j Bloom to visualize and explore our road network.

Adding Addresses

We can use data from OpenAddresses.io (https://openaddresses.io) to add addresses to our road network so that we can find routes between addresses.

Data from OpenAddresses is GeoJSON that looks like this:

```
{
  "type":"Feature",
  "properties":
    {
      "hash":"a98a4da3fa36d965",
      "number":"599",
      "street":"SKYLINE BLVD",
      "unit":"",
      "city":"DALY CITY",
      "district":"",
      "region":"",
      "postcode":" ",
      "id":"002011020"
    },
```

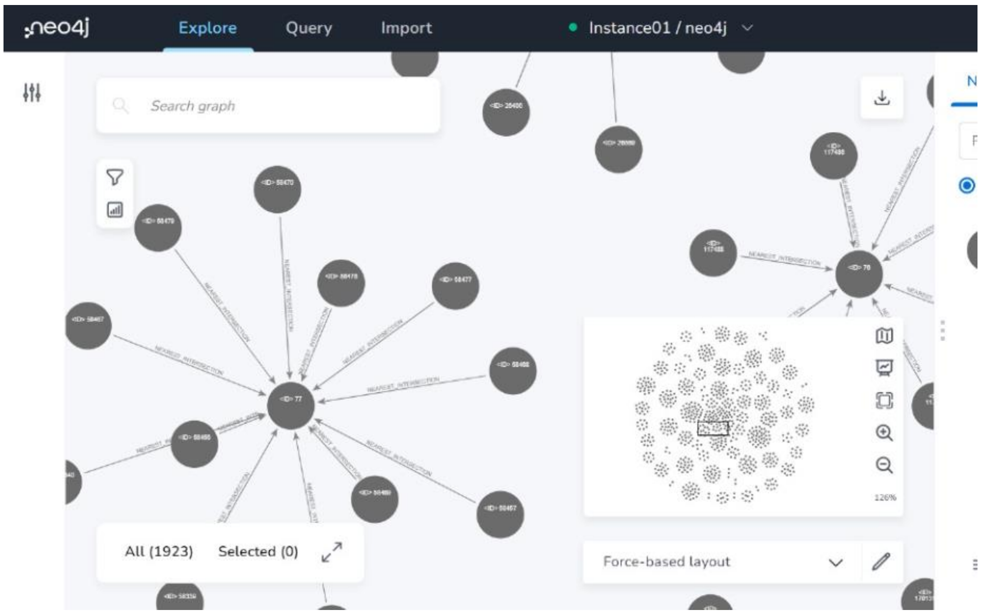
In [12]:
```
# We'll use apoc.load.json to import a JSON file of address data

add_addresses_query = """
CALL apoc.periodic.iterate(
  'CALL apoc.load.json("https://cdn.neo4jlabs.com/data/addresses/san_mateo.geo
  'MERGE (a:Address {id: value.properties.id})
SET a.location =
  point(
      {latitude: value.geometry.coordinates[1], longitude: value.geometry.coor
    a.full_address = value.properties.number + " " + value.properties.street -

SET a += value.properties',
  {batchSize:10000, parallel:true})
"""

# Next, connect each address to the road network at the nearest intersection

near_intersection_query = """
CALL apoc.periodic.iterate(
  'MATCH (p:Address) WHERE NOT EXISTS ((p)-[:NEAREST_INTERSECTION]->(:Interse
  'CALL {
  WITH p
  MATCH (i:Intersection)
  USING INDEX i:Intersection(location)
  WHERE point.distance(i.location, p.location) < 200

  WITH i
  ORDER BY point.distance(p.location, i.location) ASC
  LIMIT 1
  RETURN i
}
WITH p, i

MERGE (p)-[r:NEAREST_INTERSECTION]->(i)
SET r.length = point.distance(p.location, i.location)
RETURN COUNT(p)',
  {batchSize:1000, parallel:false})
"""

# Create a full text index to enable search in our web app

full_text_query = "CREATE FULLTEXT INDEX search_index IF NOT EXISTS FOR (p:Po
```

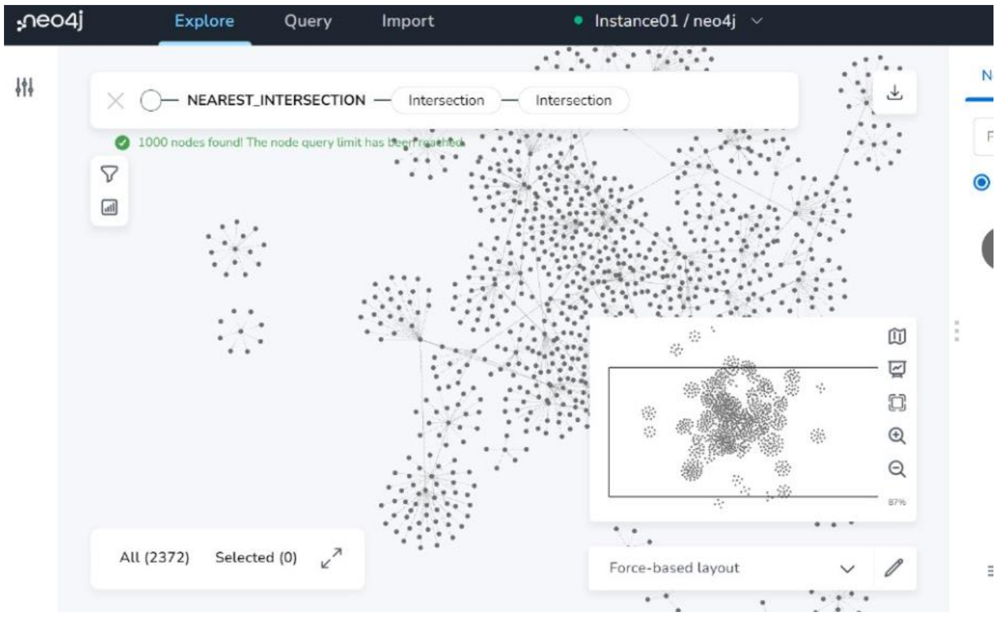
In [13]:
```
# Transaction function to execute our address import queries

def enrich_addresses(tx):
    results = tx.run(add_addresses_query)
    results = tx.run(near_intersection_query)
```

In [ ]:
```
# Execute address import
with driver.session() as session:
    session.execute_write(enrich_addresses)
```

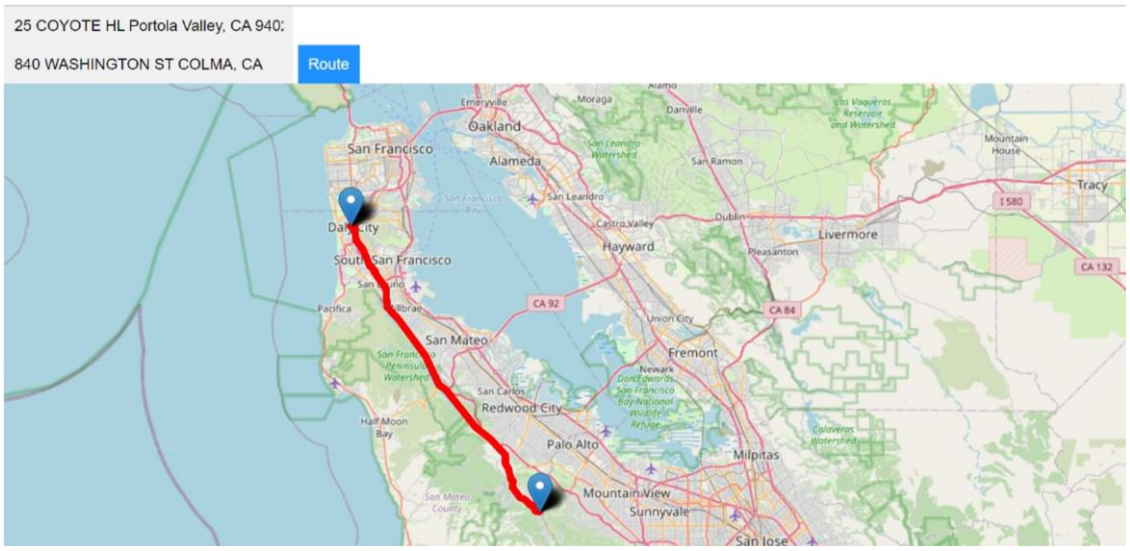
In [ ]:
```
# Execute full text index query
with driver.session() as session:
    results = session.execute_write(lambda tx: tx.run(full_text_query))
```

Output

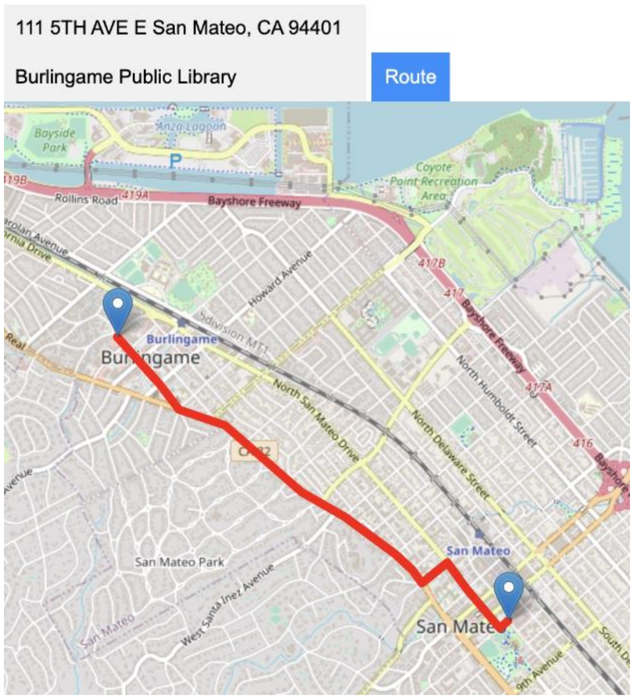




1000 nodes found! The node query limit has been reached.


25 COYOTE HL Portola Valley, CA 940:

840 WASHINGTON ST COLMA, CA


111 5TH AVE E San Mateo, CA 94401

Burlingame Public Library

Conclusion

Our Routing Web Application, powered by Neo4j, OpenStreetMap, and Leaflet.js, represents a remarkable fusion of cutting-edge technologies to address the pressing needs of efficient navigation and route optimization in our increasingly complex urban landscapes. This project has demonstrated the immense value of geospatial data analytics and graph databases in creating user-friendly, data-driven solutions. By seamlessly integrating data, enhancing user customization, and providing insightful analytics, our application offers a comprehensive toolset for both individuals and organizations.

This project underscores the potential of technology to transform the way we navigate and interact with our surroundings. As cities expand and our transportation systems evolve, the development of routing applications has never been more vital. It not only aids in minimizing environmental impact and improving the efficiency of daily travel but also serves as a foundation for the smarter, more connected cities of the future. In essence, this Routing Web Application serves as a testament to the power of innovation, data analytics, and geospatial intelligence in making our lives more convenient, sustainable, and informed.

References

[1] Modeling and Querying Trajectories using Neo4j Spatial and TimeTree for Carpool Matching 2017 IEEE

[2] A Cypher Query based NoSQL Data Mining on Protein Datasets using Neo4j Graph Database. 2017 International Conference on Advanced Computing and Communication Systems (ICACCS -2017), Jan. 06 – 07, 2017, Coimbatore, INDIA

[3] A Graph Database of Yelp Dataset Challenge 2018 and Using Cypher for Basic Statistics and Graph Pattern Exploration IEEE 2018