

# Universidad Nacional Autónoma de México

## Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

### Bases de Datos No Estructurados

### Práctica 3: MongoDB

#### Autores :

- López Sánchez Misael
- Tapia López José de Jesús.

#### Objetivos:

Desarrollar un sistema de paseos en el que:

- Un usuario se pueda dar de alta junto con información de ubicación (coordenadas), debe usar al menos un lugar (casa por ejemplo) pero con opción a agregar más (trabajo, escuela, etc.)
- El usuario debe poder buscar las estaciones de bicicleta que le queden más cercana a sus lugares.
- Debe poder planear viajes, dado un tiempo que quiere viajar, el sistema debe recomendar viajes usando como salida sus estaciones más cercanas (o estaciones específicas seleccionadas por el usuario) y los destinos que le tomen más o menos ese tiempo.
- El usuario debe poder dar la opción de que en su viaje sea redondo (mismo punto de partida y de salida). En este caso, solo se debe tomar en cuenta los datos que pasan por otras estaciones, a menos que el tiempo sea muy corto.
- Si el usuario busca una ruta específica que el sistema no tenga, la debe intentar armar de forma directa.

Extra:

- Los resultados deben tomar en cuenta la hora a la que se hace la consulta y la hora de los viajes registrados para dar sus resultados con mayor confianza.



## INTRODUCCIÓN

MongoDB es una base de datos distribuida, basada en documentos de código abierto y escrito en C++ , que de uso general que ha sido diseñada para desarrolladores de aplicaciones modernas y para la era de la nube. Al estar escrito en C++, cuenta con una más que notoria capacidad para aprovechar los recursos de la máquina y, al estar licenciado bajo una licencia GNU AGPL 3.0, es posible adaptarlo a nuestras necesidades. Además, que sea una base de datos documental significa que almacena datos en forma de documentos tipo JSON; la cual es considerada como una de las formas más naturales de concebir los datos. Mongo DB trabaja con colecciones. Una colección es muy similar a una Tabla de una base de datos. La tabla almacena registros (filas) mientras que las colecciones almacenan documentos.

## Características principales

### 1.- Consultas ad-hoc

En general, cuando diseñamos un esquema de una base de datos, no sabemos de antemano sobre las consultas que realizaremos. Las consultas ad-hoc son las consultas que no se conocen al estructurar la base de datos. Entonces, MongoDB proporciona soporte de consultas ad-hoc que lo hace tan especial en este caso. Las consultas ad-hoc se actualizan en tiempo real, lo que lleva a una mejora en el rendimiento.

### 2.- Base de datos sin esquema

En MongoDB, una colección contiene diferentes documentos. No tiene ningún esquema, por lo que puede tener muchos campos, contenidos y tamaños diferentes a los de otro documento en la misma colección. Puede tener también muchos documentos anidados. Es por eso que MongoDB muestra flexibilidad al tratar con las bases de datos.

### 3.- Orientado a documentos

MongoDB es una base de datos orientada a documentos, que es una gran característica en sí misma. En las bases de datos relacionales, hay tablas y filas para la disposición de los datos. Cada fila tiene un número específico de columnas y esas pueden almacenar un tipo específico de datos. Aquí viene la flexibilidad de

MongoDB donde hay campos en lugar de tablas y filas. Existen diferentes documentos que pueden almacenar diferentes tipos de datos. Hay colecciones de documentos similares. Cada documento tiene una identificación de clave o identificación de objeto única que puede ser definida por el usuario o por el sistema.

#### 4.- Indexación

La indexación es muy importante para mejorar el rendimiento de las consultas de búsqueda. Cuando continuamente realizamos búsquedas en un documento, debemos indexar aquellos campos que coinciden con nuestros criterios de búsqueda. En MongoDB, podemos indexar cualquier campo indexado con índices primarios y secundarios. Al realizar búsquedas más rápidas, la indexación de MongoDB mejora el rendimiento.

#### 5.- Replicación

Sabemos que la replicación es un proceso básico en la gestión de bases de datos. MongoDB soporta el tipo de replicación primario-secundario. De este modo, mientras podemos realizar consultas con el primario, el secundario actúa como réplica de datos en solo lectura a modo copia de seguridad con la particularidad de que los nodos secundarios tienen la habilidad de poder elegir un nuevo primario en caso de que el primario actual deje de responder.

#### 6.- Agregación

MongoDB tiene un marco de agregación para una usabilidad eficiente. Podemos procesar datos por lotes y obtener un único resultado incluso después de realizar diferentes operaciones en los datos del grupo. La canalización de agregación, la función de reducción de mapas y los métodos de agregación de un solo propósito son las tres formas de proporcionar un marco de agregación.

#### 7.- GridFS

GridFS es una característica de almacenamiento y recuperación de archivos. Para archivos de más de 16 MB, esta característica es muy útil. GridFS divide un documento en partes llamadas fragmentos y los almacena en un documento separado. Estos fragmentos tienen un tamaño predeterminado de 255 kb, excepto el último fragmento. Cuando consultamos a GridFS para un archivo, ensambla todos los fragmentos según sea necesario.

#### 8.- Sharding

Básicamente, el concepto de sharding viene cuando necesitamos lidiar con conjuntos de datos más grandes. Esta gran cantidad de datos puede causar algunos problemas cuando llega una consulta para ellos. Esta característica ayuda a distribuir estos datos problemáticos a múltiples instancias de MongoDB. Las colecciones en MongoDB que tiene un tamaño más grande se distribuyen en varias colecciones. Estas colecciones se llaman "fragmentos". Los fragmentos se implementan por grupos.

#### 9.- Alto rendimiento

MongoDB es una base de datos de código abierto con alto rendimiento. Esto muestra alta disponibilidad y escalabilidad. Tiene una respuesta de consulta más rápida debido a la indexación y la replicación. Esto lo convierte en una mejor opción para big data y aplicaciones en tiempo real.

## DESARROLLO

Con lo que conocemos hasta el momento de MongoDB trabajaremos con la base de datos **citibike** la cual cuenta con información de todas las estaciones de bicicleta en la ciudad de Nueva York, dicha información es sobre todos los viajes realizados en esta ciudad en bicicleta.

In [1]:

```
# Bibliotecas que utilizaremos durante la práctica
from pymongo import MongoClient
from datetime import datetime
import pymongo
import json
import numpy as np
import pandas as pd
import zipfile
import io
import requests
import random
import folium
import os
```

In [2]:

```
#Primero extraemos todos nuestros datos, trabajaremos con los de enero del 2014
data_bikes=[]

r = requests.get('https://s3.amazonaws.com/tripdata/201307-201402-citibike-tripdata')
with zipfile.ZipFile(io.BytesIO(r.content)) as ar:
    data_bikes.append(pd.read_csv(ar.open('2014-01 - Citi Bike trip data.csv')))

data_bikes=pd.concat(data_bikes)
#cheamos las columnas de los datos
data_bikes.columns
```

Out[2]:

```
Index(['tripduration', 'starttime', 'stoptime', 'start station id',
      'start station name', 'start station latitude',
      'start station longitude', 'end station id', 'end station nam
e',
      'end station latitude', 'end station longitude', 'bikeid', 'use
rtype',
      'birth year', 'gender'],
      dtype='object')
```

In [3]:

```
# Analizamos la densidad de los viajes.  
data_bikes['starttime'].map(lambda t: t.split(" ")[0]).value_counts
```

Out[3]:

```
<bound method IndexOpsMixin.value_counts of 0          2014-01-01  
1          2014-01-01  
2          2014-01-01  
3          2014-01-01  
4          2014-01-01  
5          2014-01-01  
6          2014-01-01  
7          2014-01-01  
8          2014-01-01  
9          2014-01-01  
10         2014-01-01  
11         2014-01-01  
12         2014-01-01  
13         2014-01-01  
14         2014-01-01  
15         2014-01-01  
16         2014-01-01  
17         2014-01-01  
18         2014-01-01  
19         2014-01-01  
20         2014-01-01  
21         2014-01-01  
22         2014-01-01  
23         2014-01-01  
24         2014-01-01  
25         2014-01-01  
26         2014-01-01  
27         2014-01-01  
28         2014-01-01  
29         2014-01-01  
...  
300370     2014-01-31  
300371     2014-01-31  
300372     2014-01-31  
300373     2014-01-31  
300374     2014-01-31  
300375     2014-01-31  
300376     2014-01-31  
300377     2014-01-31  
300378     2014-01-31  
300379     2014-01-31  
300380     2014-01-31  
300381     2014-01-31  
300382     2014-01-31  
300383     2014-01-31  
300384     2014-01-31  
300385     2014-01-31  
300386     2014-01-31  
300387     2014-01-31  
300388     2014-01-31  
300389     2014-01-31  
300390     2014-01-31  
300391     2014-01-31  
300392     2014-01-31
```

```

300393    2014-01-31
300394    2014-01-31
300395    2014-01-31
300396    2014-01-31
300397    2014-01-31
300398    2014-01-31
300399    2014-01-31
Name: starttime, Length: 300400, dtype: object>

```

Antes de insertar los datos en MongoDB, analizamos la demanda de la información para poder conocer donde es que la demanda de bicicletas es mayor y tener los mejores datos posibles.

En la gráfica que sigue mostramos la distribución de la demanda de viajes en bicicleta, donde podemos ver que la mayor demanda está dada en el la fecha 01/01/2014.

In [7]:

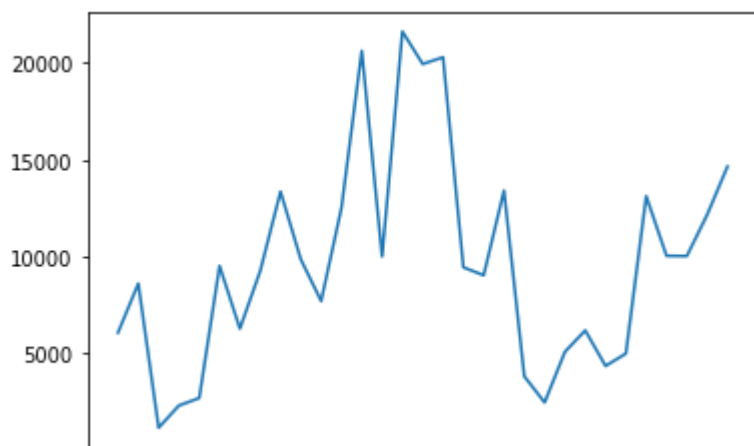
```

import matplotlib.pyplot as plt
data_bikes['starttime'].map(lambda t: t.split(" ")[0]).value_counts().sort_index()

```

Out[7]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f31a20a94a8>



In [8]:

```

# Para hacer un filtrado de los datos, parece ser 2014-01-15 es la
# fecha con más saturación de usuarios así que
# indexamos ahí
indexer = data_bikes['starttime'].map(lambda t: t.split(" ")[0] == '2014-01-15')

```

	tripduration	starttime	stoptime	start station id	start station name	start station latitude	start station longitude	end station id	end station name	end station latitude	end station longitude	bikeid	usertype	birth year	gender
119839	525	2014-01-15 00:00:02	2014-01-15 00:08:47	505	6 Ave & W 33 St	40.749013	-73.988484	469	Broadway & W 53 St	40.763441	-73.982681	20526	Subscriber	1988	1
119840	722	2014-01-15 00:00:11	2014-01-15 00:12:13	464	E 56 St & 3 Ave	40.759345	-73.967597	504	1 Ave & E 15 St	40.732219	-73.981656	15863	Subscriber	1986	2
119841	307	2014-01-15 00:00:43	2014-01-15 00:05:50	545	E 23 St & 1 Ave	40.736502	-73.978095	527	E 33 St & 1 Ave	40.743156	-73.974347	20335	Subscriber	1986	2
119842	428	2014-01-15 00:00:53	2014-01-15 00:08:01	528	2 Ave & E 31 St	40.742909	-73.977061	487	E 20 St & FDR Drive	40.733143	-73.975739	15767	Subscriber	1987	1
119843	197	2014-01-15 00:01:19	2014-01-15 00:04:36	476	E 31 St & 3 Ave	40.743943	-73.979661	518	E 39 St & 2 Ave	40.747804	-73.973442	15594	Subscriber	1983	1

In [10]:

```
# Guardamos los datos
data_bikes[indexer].to_csv("2014_indexados")
```

In [11]:

```
data_index = pd.read_csv("2014_indexados", index_col=0)
```

In [12]:

```
#Calculamos todas las estaciones de id
estaciones = set(data_bikes['start station id'].values).union(
    set(data_bikes['end station id']))
```

In [13]:

```
from tqdm import tqdm
subframes = []
start_counts = []
end_counts = []
mean_time = []

for station_id in tqdm(estaciones):
    started_here = data_bikes[data_bikes['start station id'] == station_id]
    ended_here = data_bikes[data_bikes['end station id'] == station_id]
    start_counts.append(len(data_index[data_index['start station id'] == station_id]))
    end_counts.append(len(data_index[data_index['end station id'] == station_id]))
    mean_time.append(len(data_index[data_index['tripduration'] == station_id]))
    if len(started_here) > 0:
        s = started_here.iloc[0]
        subframes.append({'latitude': s['start station latitude'],
                          'longitude': s['start station longitude'],
                          'station name': s['start station name'],
                          'station id': s['start station id'],
                          'tripduration': s['tripduration'],
                          'latitude_f': s['end station latitude'],
                          'longitude_f': s['end station longitude'],
                          'station name_f': s['end station name']})
    elif len(ended_here) > 0:
        s = ended_here.iloc[0]
        subframes.append({'latitude': s['end station latitude'],
                          'longitude': s['end station longitude'],
                          'station name': s['end station name'],
                          'station id': s['end station id'],
                          'tripduration': s['tripduration'],
                          'latitude_f': s['start station latitude'],
                          'longitude_f': s['start station longitude'],
                          'station name_f': s['start station name']})
    else:
        print("ERRORR")
```

100%|██████████| 329/329 [00:05<00:00, 58.68it/s]

In [14]:

```
stations = pd.DataFrame(subframes).set_index('station id', drop=True)
stations['incoming trips'] = start_counts
stations['outgoing trips'] = end_counts
stations['all trips'] = np.array(start_counts) + np.array(end_counts)
stations.index.name = 'station id'
del subframes
```

En la tabla siguiente podemos mostrar la manera en que está representada la demanda de bicicletas por cada estación registrada en nuestra base de datos; mostramos cómo es que están distribuidas las coordenadas de cada estación, así como el número total de viajes que llegaron, salieron y el número total de viajes por cada estación.

	latitude	latitude_f	longitude	longitude_f	station name	station name_f	tripduration	incoming trips	outgoing trips	all trips
station id										
72	40.767272	40.760193	-73.993929	-73.991255	W 52 St & 11 Ave	9 Ave & W 45 St	310	64	70	134
79	40.719116	40.711512	-74.006667	-74.015756	Franklin St & W Broadway	South End Ave & Liberty St	956	43	53	96
82	40.711174	40.742354	-74.000165	-73.989151	St James Pl & Pearl St	Broadway & W 24 St	1307	23	24	47
83	40.683826	40.682166	-73.976323	-73.953990	Atlantic Ave & Fort Greene Pl	Hancock St & Bedford Ave	660	27	29	56
116	40.741776	40.740583	-74.001497	-74.005509	W 17 St & 8 Ave	9 Ave & W 14 St	170	150	163	313
119	40.696089	40.714131	-73.978034	-73.997047	Park Ave & St Edwards St	Division St & Bowery	806	4	3	7
120	40.686768	40.696102	-73.959282	-73.967510	Lexington Ave & Classon Ave	Washington Ave & Park Ave	466	13	7	20
127	40.731724	40.740259	-74.006744	-73.984092	Barrow St & Hudson St	Lexington Ave & E 24 St	834	107	129	236
128	40.727103	40.736502	-74.002971	-73.978095	MacDougal St & Prince St	E 23 St & 1 Ave	1747	135	149	284
137	40.761628	40.756014	-73.972924	-73.967416	E 56 St & Madison Ave	E 52 St & 2 Ave	390	72	74	146
143	40.692395	40.697601	-73.993379	-73.993446	Clinton St & Joralemon St	Clark St & Henry St	238	14	20	34
144	40.698399	40.725029	-73.980689	-73.990697	Nassau St & Navy St	E 2 St & 2 Ave	1158	4	5	9
146	40.716250	40.739126	-74.009106	-73.979738	Hudson St & Reade St	E 25 St & 2 Ave	1619	48	43	91
147	40.715422	40.711512	-74.011220	-74.015756	Greenwich St & Warren St	South End Ave & Liberty St	326	84	83	167
150	40.720874	40.727408	-73.980858	-73.981420	E 2 St & Avenue C	E 10 St & Avenue A	264	50	44	94

In [16]:

```
# Cantidad total de estaciones contabilizadas.
len(stations)
```

Out[16]:

329

In [17]:

```
# ¿Cuántas estaciones hay en el sistema?, Coinciden!!
# así que sí calculamos bien todas las estaciones disponibles
len(data_bikes['start station name'].unique())
```

Out[17]:

329

In [18]:

```
stations.to_csv("amazon_tripdata")
```

Realizamos la conexión entre MongoDB de manera remota ayudandonos de la plataforma **fastdl.mongodb** y



comenzamos a insertar nuestros registros.

In [19]:

```
# MongoDB download and installation
# Downloads MongoDB from official repository
!wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-debian71-3.0.15.tgz
# Unpack compressed file
!tar xfv mongodb-linux-x86_64-debian71-3.0.15.tgz >/dev/null
# Removes downloaded file
!rm mongodb-linux-x86_64-debian71-3.0.15.tgz
```

```
--2020-05-02 17:18:48-- https://fastdl.mongodb.org/linux/mongodb-linu
x-x86_64-debian71-3.0.15.tgz (https://fastdl.mongodb.org/linux/mongodb
-linux-x86_64-debian71-3.0.15.tgz)
Resolving fastdl.mongodb.org (fastdl.mongodb.org)... 2600:9000:211f:4c
00:5:c2a6:d640:93a1, 2600:9000:211f:9e00:5:c2a6:d640:93a1, 2600:9000:2
11f:9c00:5:c2a6:d640:93a1, ...
Connecting to fastdl.mongodb.org (fastdl.mongodb.org)|2600:9000:211f:4
c00:5:c2a6:d640:93a1|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 70878938 (68M) [application/x-gzip]
Saving to: 'mongodb-linux-x86_64-debian71-3.0.15.tgz'
```

```
mongodb-linux-x86_64 100%[=====>] 67,59M 1,07MB/s in
73s
```

```
2020-05-02 17:20:02 (943 KB/s) - 'mongodb-linux-x86_64-debian71-3.0.1
5.tgz' saved [70878938/70878938]
```

Realizamos la conexión con MongoDB y creamos nuestra primera base de datos.

In [20]:

```
uri = 'mongodb://ulkkdrchfjim80tclysv:FeesC2ACNmI7be61RTst@brny4kjelauboxl-mongodb.
+27017/brny4kjelauboxl'

client = MongoClient( uri )
```

In [21]:

```
db = client.brny4kjelauboxl
collection = db.bicing # Coleccion alias
```

In [22]:

```
db.list_collection_names()
```

Out[22]:

```
['mapas', 'addressbook', 'bicing', 'practica', 'videogames']
```

In [23]:

```
bd_bikes = pd.read_csv("amazon_tripdata")
```

Como se nos pide en la práctica, realizamos la extracción de los datos de cada una de las estaciones y las almacenamos dentro de MongoDB usando el formato de geojson especificado, haciendo uso de índices para

guardar nuestra información.

In [24]:

```
# Le cambiamos el id de las estaciones para un mejor control de estas

for i in range(len(bd_bikes)):
    bd_bikes['station id'][i] = i
bd_bikes.head()
```

/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:4: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)  
after removing the cwd from sys.path.

Out[24]:

	station id	latitude	latitude_f	longitude	longitude_f	station name	station name_f	tripduration	inc
0	0	40.767272	40.760193	-73.993929	-73.991255	W 52 St & 11 Ave	9 Ave & W 45 St	310	
1	1	40.719116	40.711512	-74.006667	-74.015756	Franklin St & W Broadway	South End Ave & Liberty St	956	
2	2	40.711174	40.742354	-74.000165	-73.989151	St James Pl & Pearl St	Broadway & W 24 St	1307	
3	3	40.683826	40.682166	-73.976323	-73.953990	Atlantic Ave & Fort Greene Pl	Hancock St & Bedford Ave	660	
4	4	40.741776	40.740583	-74.001497	-74.005509	W 17 St & 8 Ave	9 Ave & W 14 St	170	

In [25]:

```
len(bd_bikes)
```

Out[25]:

329

In [26]:

```
# Cuatro listas, cada una tiene las coordenadas de las estaciones de las bicis
bike_station_latitude = []
bike_station_longitude = []
bike_station_latitude_f = []
bike_station_longitude_f = []

for i in range(len(bd_bikes)):
    bike_station_latitude.append(bd_bikes['latitude'][i])
    bike_station_longitude.append(bd_bikes['longitude'][i])
    bike_station_latitude_f.append(bd_bikes['latitude_f'][i])
    bike_station_longitude_f.append(bd_bikes['longitude_f'][i])
```

In [27]:

```
# Esto lo hacemos para guardar el dataframe en Mongo, primero guardamos
# todos los atributos que no son de coordenadas, y después guardaremos
# las que son coordenadas, ya que estas últimas requieren un proceso
# especial para las futuras consultas que nos pide la práctica

bd2_bikes = bd_bikes.copy()
```

In [28]:

```
bd2_bikes = bd2_bikes.drop(['latitude', 'longitude', 'latitude_f', 'longitude_f'],
                           axis = 1)
bd2_bikes.head()
```

Out[28]:

	station id	station name	station name_f	tripduration	incoming trips	outgoing trips	all trips
0	0	W 52 St & 11 Ave	9 Ave & W 45 St	310	64	70	134
1	1	Franklin St & W Broadway	South End Ave & Liberty St	956	43	53	96
2	2	St James Pl & Pearl St	Broadway & W 24 St	1307	23	24	47
3	3	Atlantic Ave & Fort Greene Pl	Hancock St & Bedford Ave	660	27	29	56
4	4	W 17 St & 8 Ave	9 Ave & W 14 St	170	150	163	313

In [29]:

```
# En esta parte es en la que agregamos el dataframe a mongo,
# usando la coleccion 'collection'
# O sea, a 'collection' le agrego el dataframe

# Eliminamos cada que ejecutamos para no tener información redundante
collection.drop()
datos_dict = bd2_bikes.to_dict("records") # Insertamos en la coleccion
collection.insert_many(datos_dict)
```

Out[29]:

<pymongo.results.InsertManyResult at 0x7f31a24aef48>

In [30]:

```
for i in range(len(bd2_bikes)):
    collection.update_one({"station id": i},
{"$set": {'start station coordinate':
{'type': "Point", 'coordinates': [bike_station_longitude[i],
                                   bike_station_latitude[i]]}}})
    collection.update_one({"station id": i},
{"$set": {'end station coordinate':
{'type': "Point", 'coordinates': [bike_station_longitude_f[i],
                                   bike_station_latitude_f[i]]}}})
```

In [31]:

```
# Le creamos el índice a las coordenadas de las dos estaciones
collection.create_index([('start station coordinate',pymongo.GEOSPHERE)])
collection.create_index([('end station coordinate',pymongo.GEOSPHERE)])
```

Out[31]:

```
'end station coordinate_2dsphere'
```

In [32]:

```
# Verificamos que se hayan guardado
```

```
for doc in collection.find().limit(5):  
    print(doc)
```

```
{'_id': ObjectId('5eadf266db3d6b1a969903b6'), 'station id': 0, 'station name': 'W 52 St & 11 Ave', 'station name_f': '9 Ave & W 45 St', 'tripduration': 310, 'incoming trips': 64, 'outgoing trips': 70, 'all trips': 134, 'start station coordinate': {'type': 'Point', 'coordinates': [-73.99392888, 40.76727216]}, 'end station coordinate': {'type': 'Point', 'coordinates': [-73.9912551, 40.76019252]}}  
{'_id': ObjectId('5eadf266db3d6b1a969903b7'), 'station id': 1, 'station name': 'Franklin St & W Broadway', 'station name_f': 'South End Ave & Liberty St', 'tripduration': 956, 'incoming trips': 43, 'outgoing trips': 53, 'all trips': 96, 'start station coordinate': {'type': 'Point', 'coordinates': [-74.00666661, 40.71911552]}, 'end station coordinate': {'type': 'Point', 'coordinates': [-74.01575600000002, 40.711512]}}  
{'_id': ObjectId('5eadf266db3d6b1a969903b8'), 'station id': 2, 'station name': 'St James Pl & Pearl St', 'station name_f': 'Broadway & W 24 St', 'tripduration': 1307, 'incoming trips': 23, 'outgoing trips': 24, 'all trips': 47, 'start station coordinate': {'type': 'Point', 'coordinates': [-74.00016545, 40.71117416]}, 'end station coordinate': {'type': 'Point', 'coordinates': [-73.98915076, 40.7423543]}}  
{'_id': ObjectId('5eadf266db3d6b1a969903b9'), 'station id': 3, 'station name': 'Atlantic Ave & Fort Greene Pl', 'station name_f': 'Hancock St & Bedford Ave', 'tripduration': 660, 'incoming trips': 27, 'outgoing trips': 29, 'all trips': 56, 'start station coordinate': {'type': 'Point', 'coordinates': [-73.97632328, 40.68382604]}, 'end station coordinate': {'type': 'Point', 'coordinates': [-73.95399026, 40.68216564]}}  
{'_id': ObjectId('5eadf266db3d6b1a969903ba'), 'station id': 4, 'station name': 'W 17 St & 8 Ave', 'station name_f': '9 Ave & W 14 St', 'tripduration': 170, 'incoming trips': 150, 'outgoing trips': 163, 'all trips': 313, 'start station coordinate': {'type': 'Point', 'coordinates': [-74.00149746, 40.74177603]}, 'end station coordinate': {'type': 'Point', 'coordinates': [-74.00550867, 40.7405826]}}
```

In [33]:

```
# Contamos los documentos base de la coleccion 'collection'
num_documents = collection.count_documents({'_id' : {'$exists' : 1}})
print ( 'Number of documents in database = ' + str(num_documents) )
list ( collection.find().limit(1) )
```

Number of documents in database = 329

Out[33]:

```
[{'_id': ObjectId('5eadf266db3d6b1a969903b6'),
  'station id': 0,
  'station name': 'W 52 St & 11 Ave',
  'station name_f': '9 Ave & W 45 St',
  'tripduration': 310,
  'incoming trips': 64,
  'outgoing trips': 70,
  'all trips': 134,
  'start station coordinate': {'type': 'Point',
  'coordinates': [-73.99392888, 40.76727216]},
  'end station coordinate': {'type': 'Point',
  'coordinates': [-73.9912551, 40.76019252]}}]
```

Por lo tanto, podemos notar que esta base de datos en mongo contiene información de las estaciones de las bicicletas y de algunos viajes que se realizan entre estas; la cual nos servirá para realizar las consultas que pide la práctica.

Creemos una segunda base de datos para poder almacenar la información de las coordenadas de cada una de las estaciones de bicicletas, lo anterior con la intención de que estas nos puedan ser de utilizadas al momento de visualizar con **folium** en un mapa interactivo cada una de las estaciones cercanas con respecto a cada una de nuestras consultas.

In [34]:

```
# Creamos otra colección para guardar los datos del data frame stations
# y usarlos con folium
db = client.brny4kjelauboxl
coleccionMaps = db.mapas
```

In [35]:

```
# Insertamos los datos de stations
coleccionMaps.drop()
datos_mapas = stations.to_dict("records")

coleccionMaps.insert_many(datos_mapas)
```

Out[35]:

<pymongo.results.InsertManyResult at 0x7f31a29d2388>

In [36]:

```
# Contamos los documentos base de la coleccion 'collection'
num_documents = coleccionMaps.count_documents({'_id' : {'$exists' : 1}})
print ( 'Number of documents in database = ' + str(num_documents) )
list ( coleccionMaps.find().limit(2) )
```

Number of documents in database = 329

Out[36]:

```
[{'_id': ObjectId('5eadf347db3d6b1a969904ff'),
  'latitude': 40.76727216,
  'latitude_f': 40.76019252,
  'longitude': -73.99392888,
  'longitude_f': -73.9912551,
  'station name': 'W 52 St & 11 Ave',
  'station name_f': '9 Ave & W 45 St',
  'tripduration': 310,
  'incoming trips': 64,
  'outgoing trips': 70,
  'all trips': 134},
 {'_id': ObjectId('5eadf347db3d6b1a96990500'),
  'latitude': 40.71911552,
  'latitude_f': 40.711512,
  'longitude': -74.00666661,
  'longitude_f': -74.015756000000001,
  'station name': 'Franklin St & W Broadway',
  'station name_f': 'South End Ave & Liberty St',
  'tripduration': 956,
  'incoming trips': 43,
  'outgoing trips': 53,
  'all trips': 96}]
```

Creemos una tercera base de datos la cual nos servira como auxiliar para poder insertar usuarios con ubicaciones dispersas a lo largo de la ciudad. Estas nos seran de utilidad al momento de realizar visualización de la información almaenada en MongoDB

In [37]:

```
# BD en Mongo que nos va a servir para las consultas
db = client.brny4kjelauboxl
coleccion = db.practica          # Collection alias
```

Notemos que las bases de datos creadas anteriormente e insertadas en Mongo, contienen la misma información de las estaciones de las bicicletas. Por lo tanto, las próximas bases de datos contendrán información del usuario.

La siguiente celda solamente la mantenemos para mostrar cómo es que generamos la base de datos que insertaremos en una coleccion de mongo, nombrada 'coleccion'.

In [38]:

```
"""
# Subimos a google colab el CSV que contiene el nombre de 1000 usuarios
# los nombres de los usuarios los generamos con https://www.mockaroo.com/

from google.colab import files

usuarios = files.upload()

# Leemos los datos

# mongo_users era el csv que solamente contenia el id y el nombre del usuario
(y el que generamos con https://www.mockaroo.com/)
usuarios = pd.read_csv("mongo_users.csv")

# Le modificamos los id's a cada usuario solamente para que no haya confusion
con el id
# la BD de la bici, pues son distintos

for i in range(len(usuarios)):
    usuarios['id'][i]=usuarios['id'][i] + 1000

# Para saber el rango en el que se encuentra las latitudes y longitudes
y ponerle aleatoriamente coordenadas de sus casas
# a los usuarios

lat_min = min(bike_station_latitude)
lat_max = max(bike_station_latitude)

long_min = min(bike_station_longitude)
long_max = max(bike_station_longitude)

# Llenamos estas nuevas consultas, que posteriormente vamos a ponerle las
coordenadas de sus casas aleatoriamente
usuarios['house latitude'] = 0
usuarios['house longitude'] = 0

usuarios['house latitude'] = usuarios['house latitude'].astype(float)
usuarios['house longitude'] = usuarios['house longitude'].astype(float)

# Aleatoriamente, le agregamos coordenadas a sus casas

for i in range(len(usuarios)):
    usuarios['house latitude'][i] = float(random.uniform(lat_min, lat_max))
    usuarios['house longitude'][i] = float(random.uniform(long_min, long_max))

# Nos montamos a google drive para exportar el dataframe que acabamos de generar
# y es el cual usaremos para la coleccion 'coleccion'
from google.colab import drive
drive.mount('drive')

# lo exportamos y lo guardaremos en mi google drive

usuarios.to_csv('drive/My Drive/usuarios_mongo.csv')
"""
```

Out[38]:



```
'\n\n# Subimos a google colab el CSV que contiene el nombre de 1000 usuarios\n# los nombres de los usuarios los generamos con https://www.mockaroo.com/\n\nfrom (https://www.mockaroo.com/\n\nfrom) google.colab import files\n\nusuarios = files.upload()\n\n# Leemos los datos\n\n# mongo_users era el csv que solamente contenia el id y el nombre del usuario (y el que generamos con https://www.mockaroo.com/)\n\nusuarios (https://www.mockaroo.com/)\n\nusuarios) = pd.read_csv("mongo_users.csv")\n\n# Le modificamos los id's a cada usuario solamente para que no haya confusion con el id\n# la BD de la bici, pues son distintos\n\nfor i in range(len(usuarios)):\n    usuarios['id'][i]=usuarios['id'][i] + 1000\n\n# Para saber el rango en el que se encuentra las latitudes y longitudes y ponerle aleatoriamente coordenadas de sus casas\n# a los usuarios\n\nlat_min = min(bike_station_latitude)\nlat_max = max(bike_station_latitude)\n\nlong_min = min(bike_station_longitude)\nlong_max = max(bike_station_longitude)\n\n# Llenamos estas nuevas consultas, que posteriormente vamos a ponerle las coordenadas de sus casas aleatoriamente\n\nusuarios['house_latitude'] = 0\nusuarios['house_longitude'] = 0\n\nusuarios['house_latitude'] = usuarios['house_latitude'].astype(float)\nusuarios['house_longitude'] = usuarios['house_longitude'].astype(float)\n\n# Aleatoriamente, le agregamos coordenadas a sus casas\n\nfor i in range(len(usuarios)):\n    usuarios['house_latitude'][i] = float(random.uniform(lat_min, lat_max))\n    usuarios['house_longitude'][i] = float(random.uniform(long_min, long_max))\n\n# Nos montamos a google drive para exportar el dataframe que acabamos de generar\n# y es el cual usaremos para la coleccion 'coleccion'\n\nfrom google.colab import drive\ndrive.mount('drive')\n\n# lo exportamos y lo guardaremos en mi google drive\n\nusuarios.to_csv('drive/My Drive/usuarios_mongo.csv')\n'
```

In [40]:

```
# Ahora sí, leemos el archivo que generamos anteriormente:
#from google.colab import files
#files.upload()
```

In [41]:

```
# leemos los datos y mostramos las primeras cinco filas
usuarios =pd.read_csv("usuarios_mongo.csv")
usuarios = usuarios.drop(['Unnamed: 0'],axis=1)
usuarios.head()
```

Out[41]:

	id	user_name	house latitude	house longitude
0	1001	Norby Maton	40.756616	-73.957374
1	1002	Winthrop Lawson	40.716022	-73.973583
2	1003	Claretta Spilstead	40.748127	-73.978596
3	1004	Tarrance Malinson	40.743398	-73.977825
4	1005	Timi Blazewski	40.735241	-73.975718

In [42]:

```
# De este ultimo dataframe, subimos las dos primeras columnas a mongo,
# pero lo hacemos en otra coleccion
# las otras dos columnas las guardo de manera conjunta, pues para la práctica,
# en mongo deben de tener
# características especiales, como lo vimos en clase

# En esta parte es en la que agregamos el dataframe a mongo, usando
# la coleccion 'coleccion'
# O sea, a 'coleccion' le agrego el dataframe

coleccion.drop()
# Insertamos en la coleccion
datos_dict = usuarios[['id', 'user_name']].to_dict("records")
coleccion.insert_many(datos_dict)
```

Out[42]:

<pymongo.results.InsertManyResult at 0x7f31a90401c8>

In [43]:

```
# Dos listas, cada una tiene sus coordenadas
house_latitude = []
house_longitude = []

for i in range(len(usuarios)):
    house_latitude.append(usuarios['house latitude'][i])
    house_longitude.append(usuarios['house longitude'][i])
```

In [44]:

```
# Ahora sí, insertamos las coordenadas de la casa de cada usuario
# a la coleccion 'coleccion'

for i in range(len(usuarios)):
    coleccion.update_one({"id": i+1001}, {"$set":
{'house coordinate': { 'type': "Point", 'coordinates':
    [house_longitude[i],house_latitude[i]]}}})
```

In [45]:

```
# Le agregamos el indice
coleccion.create_index([('house coordinate', pymongo.GEOSPHERE)])

# Verificamos que se hayan guardado

for doc in coleccion.find().limit(5):
    print(doc)

{'_id': ObjectId('5eadf36cdb3d6b1a96990648'), 'id': 1001, 'user_name':
'Norby Maton', 'house coordinate': {'type': 'Point', 'coordinates': [-
73.95737408095505, 40.75661568772006]}}
{'_id': ObjectId('5eadf36cdb3d6b1a96990649'), 'id': 1002, 'user_name':
'Winthrop Lawson', 'house coordinate': {'type': 'Point', 'coordinate
s': [-73.9735826207915, 40.716022026719095]}}
{'_id': ObjectId('5eadf36cdb3d6b1a9699064a'), 'id': 1003, 'user_name':
'Claretta Spilstead', 'house coordinate': {'type': 'Point', 'coordinat
es': [-73.97859580579158, 40.74812720998217]}}
{'_id': ObjectId('5eadf36cdb3d6b1a9699064b'), 'id': 1004, 'user_name':
'Tarrance Malinson', 'house coordinate': {'type': 'Point', 'coordinate
s': [-73.97782518094208, 40.743397509327046]}}
{'_id': ObjectId('5eadf36cdb3d6b1a9699064c'), 'id': 1005, 'user_name':
'Timi Blazewski', 'house coordinate': {'type': 'Point', 'coordinates':
[-73.97571831319989, 40.73524062342115]}}
```

In [46]:

```
# Contamos los documentos base de la coleccion 'coleccion'
num_documents = coleccion.count_documents({'_id' : {'$exists' : 1}})
print ( 'Number of documents in database = ' + str(num_documents) )
list ( coleccion.find().limit(1) )
```

Number of documents in database = 1000

Out[46]:

```
[{'_id': ObjectId('5eadf36cdb3d6b1a96990648'),
'id': 1001,
'user_name': 'Norby Maton',
'house coordinate': {'type': 'Point',
'coordinates': [-73.95737408095505, 40.75661568772006]}}]
```

In [47]:

```
usuario_indx = coleccion.find({'id':{'$exists':True}}, ['coordinates']).limit(1)
print(usuario_indx)
```

<pymongo.cursor.Cursor object at 0x7f31a9048198>

## Aplicación:

La siguiente es la aplicación principal de nuestro sistema de bicicletas implementado en \*MongoDB\*. Cuenta con un menú con las siguientes opciones:

### 1.- Ingresar nuevo usuario -

Se pueden dar de alta a nuevos usuarios dentro de la base de datos, y al igual que con los demás documentos, inicialmente solo se le pide la ubicación de su casa; pero claramente en la otra opción puede dar de alta otras ubicaciones.

## **2.- Ingresar nueva ubicación de usuario -**

Los usuarios pueden ingresar nuevas direcciones que visitan frecuentemente,, por ejemplo: gimnasio, escuela, trabajo, etc.

## **3.- Localizar módulo(s) más cercano(s) -**

Le devuelve al usuario el número y el nombre de estaciones de bicicletas más cercanas que él indique, con respecto al nombre de la ubicación que ingresen.

## **4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar-**

El usuario indica el tiempo aproximado que quiere viajar y se le muestran posibles rutas de bicicleta. Él puede indicar si el inicio de su viaje es desde la(s) estación(es) de bicicleta(s) cercana(s) al nombre de su ubicación que sugiere, o indicando posibles estaciones por las que les gustaría que sean las salidas de su viaje.

## **5.- Viaje redondo -**

Se le muestra la opción de un viaje redondo con escalas especificadas por el usuario, en diferentes estaciones.

## **6.- Recomendaciones de rutas -**

De acuerdo a un tiempo de viaje mínimo entre estaciones de bicicleta,el sistema le devuelve al usuario un circuito completo de viaje.

## **7.- Crea tu propia ruta -**

El usuario es capaz de definir su propia ruta, indicando las estaciones de bicicleta.

## **8.- Salir -**

Fin de la Aplicación.

Cabe destacar que el usuario puede visualizar en un mapa el resultado de seleccionar alguna de las siguientes opciones: 3,4,5,6 o 7.

In [ ]:

```
# Aplicación
```

```
os.system('cls')
clear = lambda:os.system('cls')
```

```
# Definimos el menú de nuestra aplicación
```

```
def menu_aplication():

    print("")
    print('+ '*20, " MENÚ ", '+'*20, '\n')
    print("1.- Ingresar nuevo usuario")
    print("2.- Ingresar una nueva ubicación a un usuario")
    print("3.- Localizar módulo(s) más cercano(s)")
    print("4.- Planear viajes, indicando el tiempo aproximado en el que se viaja")
    print("5.- Viaje redondo")
    print("6.- Recomendaciones de rutas")
    print("7.- Crea tu propia ruta")
    print("8.- Salir")
    print("")
```

```
# Iniciamos la interfaz grafica con todo su desarrollo
```

```
while True:
    menu_aplication()
    opMenu = input("Ingrese una opción >> ")
    clear()

    if opMenu == '1':
        print('- '*5, " Ingresar datos de usuario ", '- '*5, '\n')
        nombre = str(input("Introduzca el nombre completo del usuario nuevo: "))

        print("Coordenadas de su casa")
        latitude = float(input("Introduzca la latitud de su casa: "))
        longitude = float(input("Introduzca la longitud de su casa: "))

        # Agregamos el usuario con el id que le sigue(id después del último documento)
        coleccion.insert_one({"id": coleccion.count_documents({'_id' :
                                                                    {'$exists' : 1}})
                              +1001, 'user_name':nombre, 'house coordinate': { 'type': "Point", 'coordinates':
                                                                              [longitude,latitude]}}})

    elif opMenu == '2':
        print('- '*5, " Ingresar una nueva ubicación a un usuario ", '- '*5, '\n')
        identificador = int(input("Introduzca el id del usuario: "))
        print("Coordenadas de la nueva ubicación")
        ubicacion = str(input("Introduzca el nombre de la nueva ubicación: "))
        ubicacion = ubicacion + " coordinate"
        latitude = float(input("Introduzca la latitud de la nueva ubicación: "))
        longitude = float(input("Introduzca la longitud de la nueva ubicación: "))

        coleccion.update_one({"id":identificador}, {"$set": {ubicacion:
                                                              { 'type': "Point", 'coordinates': [longitude,latitude]}}})

    elif opMenu == '3':
        print('- '*5, " Localizar módulo más cercano ", '- '*5, '\n')

        identificador = int(input("Introduzca el ID del usuario: "))
        ubicacion = str(input("Introduzca el nombre de la ubicación: "))
        loc = ubicacion + " coordinate"
```

```

# Obtenemos las coordenadas de la ubicacion proporcionada por el usuario
for doc in coleccion.find({'id':identificador}):
    coord = doc
    coordenadas = coord[loc]['coordinates']

center_lat = coordenadas[1]
center_lon = coordenadas[0]

# inicializamos mapa en la posicion de nuestro usuario
locationmap = folium.Map(location=[ center_lat , center_lon ],
                          zoom_start=14, width=1800, height=600 )

# con flecha azul le indicamos al usuario donde se encuentra
folium.Marker(location=[center_lat,center_lon],
              popup='Bienvenido <b> Usted se encuentra aqui</b>').add_to(locationmap)

consulta = {"start station coordinate": {"$near":
    {"$geometry": {"type":"Point", "coordinates":coordenadas },
    "$maxDistance": 2000}}}

# Esta lista vacía nos permitirá guardar el nombre de las estaciones
# de bicis más cercanas
estaciones = []

cantidad_estaciones = int(
input("Introduzca la cantidad de estaciones de bicicleta que desea obtener:"))
print("El nombre de la(s) estacion(es) más cercana(s)
+ "(de menor a mayor distancia) \n a la ubicación que proporcionó es(son): ")
# Notemos que la consulta la hacemos sobre la BD de las bicis
for doc in collection.find(consulta,
                          {'station name':1}).limit(cantidad_estaciones):
    print("-"+doc['station name'])
    # Guardamos en la lista el nombre de las estaciones de bicis más cercanas
    estaciones.append(doc['station name'])
estaciones_consulta = []
for i in range(len(estaciones)):
    estaciones_consulta.append({'station name':estaciones[i]})

# En el siguiente filtro es para quedarnos con los documentos que
# tengan las estaciones más cercanas
filters = {"$or": estaciones_consulta}
fields = { '_id', 'latitude', 'longitude', 'station name', 'tripduration',
            'latitude_f', 'longitude_f', 'station name_f' }

#realizamos la consulta para obtener las coordenadas
#de las estaciones más cercanas
query = list( coleccionMaps.find( filters , fields ) )
df = pd.DataFrame ( query )

print(df[['station name']])
#Dibujamos todas la(s) posible(s) trayectoria(s) que el usuario puede tomar
for _, row in df.iterrows():
    folium.CircleMarker([row['latitude'], row['longitude']],
                        radius=15,
                        fill_color="green", # divy color
                        ).add_to(locationmap)

display(locationmap)

elif opMenu == '4':
    print('-'*5," Planear viajes, indicando el tiempo aproximado en

```

```

        + "el que se quiere viajar ", '- '*5, '\n')
print("1.- Planear viajes usando como posible salida las estaciones
      + "más cercanas de la ubicación del usuario ")
print("2.- Planear viajes usando como posible salida las estaciones
      + "que usted nos proporcione ")
print()

opcion = input("Introduzca una opción: ")

# Esta lista vacía nos permitirá guardar el nombre de
# las estaciones de bicis
estaciones = []

if opcion == '1':

    identificador = int(input("Introduzca el ID del usuario: "))
    ubicacion = str(input("Introduzca el nombre de la ubicación: "))
    loc = ubicacion + " coordinate"
    # Obtenemos las coordenadas de la ubicacion
    # proporcionada por el usuario
    for doc in coleccion.find({'id': identificador}):
        coord = doc
        coordenadas = coord[loc]['coordinates']

        center_lat = coordenadas[1]
        center_lon = coordenadas[0]

        # inicializamos mapa en la posicion de nuestro usuario
        locationmap = folium.Map(location=[center_lat, center_lon],
                                   zoom_start=14, width=1800, height=600)

        # con flecha azul le indicamos al usuario donde se encuentra
        folium.Marker(location=[center_lat, center_lon],
                      popup='Bienvenido <b> Usted se encuentra aqui</b>').add_to(locationmap)

        # Hacemos la consulta para saber las estaciones de bicis más cercanas
        # a la ubicación del usuario
        consulta = {"start station coordinate":
                    {"$near": {"$geometry": {"type": "Point", "coordinates": coordenadas },
                              "$maxDistance": 2000}}}

        cantidad_estaciones = int(input("Introduzca la cantidad de estaciones
                                          + "de bicicleta que desearía obtener: "))
        print("El nombre de la(s) estacion(es) más cercana(s)
              + "(de menor a mayor distancia)
              + "\n a la ubicación que proporcionó es(son): ")
        # Notemos que la consulta la hacemos sobre la BD de las bicis
        for doc in collection.find(consulta,
                                   {'station name': 1}).limit(cantidad_estaciones)
            print("-"+doc['station name'])
        # Guardamos en la lista el nombre de las estaciones de bicis más cercanas
        estaciones.append(doc['station name'])

elif opcion == '2':
    cantidad_estaciones = int(input("Introduzca la cantidad de estaciones
    + "de bicicleta que desearía obtener como posible ruta: "))
    print("Introduzca el nombre de las estaciones: ")
    for i in range(cantidad_estaciones):
        estacion = str(input("Introduzca el nombre de la estación {0}: ".format(i+1)))
        estaciones.append(estacion)

```

```

estaciones_consulta = []
for i in range(len(estaciones)):
    estaciones_consulta.append({'station name':estaciones[i]})

tViaje = int(input("Introduzca el tiempo aproximado (en segundos)
                    + "que quiere que dure su viaje: "))

# En el siguiente filtro es para quedarnos con los
# documentos que tengan las
# estaciones correspondientes y que
# se aproximen al tiempo proporcionado (Podemos notar que en ambas opciones
# no todas las estaciones pueden
# cumplir el criterio del tiempo aproximado)
filters = {'tripduration' : {"$gte": tViaje-60, "$lte": tViaje+60},
          "$or": estaciones_consulta}
fields = { '_id', 'latitude', 'longitude', 'station name',
          'tripduration', 'latitude_f', 'longitude_f', 'station name_f'}

# realizamos una consulta de todos los viajes que duren aproximadamente
# el tiempo especificado por el usuario
query = list( coleccionMaps.find( filters , fields ) )
print("De las estaciones obtenidas como inicio del viaje,
+ "las que tienen una ruta con un tiempo aproximado al proporcionado son: ")
df = pd.DataFrame ( query )

print(df[['station name', 'tripduration', 'station name_f']])
#Dibujamos todas la(s) posible(s) trayectoria(s) que el
# usuario puede tomar
for _, row in df.iterrows():
    folium.CircleMarker([row['latitude'], row['longitude']],
                        radius=15,
                        fill_color="green", # divy color
                        ).add_to(locationmap)

    folium.CircleMarker([row['latitude_f'], row['longitude_f']],
                        radius=15,
                        fill_color="red", # divy color
                        ).add_to(locationmap)

    folium.PolyLine([[row['latitude'], row['longitude']],
                    [row['latitude_f'],
                     row['longitude_f']]]).add_to(locationmap)

display(locationmap)

elif opMenu == '5':
    print('-'*5, " Viaje Redondo ", '-'*5, '\n')
    misEscalas_cerca = []

    identificador = int(input("Introduzca el ID del usuario: "))
    ubicacion = str(input("Introduzca el nombre de la ubicación: "))
    loc = ubicacion + " coordinate"
    # Obtenemos las coordenadas de la ubicacion proporcionada por el usuario
    for doc in coleccion.find({'id': identificador}):
        coord = doc
        coordenadas = coord[loc]['coordinates']

    center_lat = coordenadas[1]
    center_lon = coordenadas[0]

```



```

consulta = {"start station coordinate": {"$near": {"$geometry":
{"type":"Point", "coordinates":coordenadas }, "$maxDistance": 2000}

# Notemos que la consulta la hacemos sobre la BD de las bicis
for doc in collection.find(consulta,{'station name':1}).limit(1):
    print("El nombre de la estación más cercana a la ubicación
        + "que proporcionó es:",
            doc['station name'])
estacionop1 = doc['station name']
# Le damos al usuario su estación más cercana
estacion1 = stations.loc[stations.loc[:, 'station name'] == estacionop1]
misEscalas_cerca.append(estacion1)
# Iniciamos el proceso de definir escalas
nEscalas = int(input("¿A cuántos de sus lugares registrados deseas ir?"))
#Introduce otro lugar estipulado
for i in range(nEscalas):
    #ident = int(input("Introduzca el ID del usuario: "))
    ubicacion = str(input("Introduzca el nombre de la ubicación {0}: "
        .format(i+1)))
    loc = ubicacion + " coordinate"
    # Obtenemos las coordenadas de la ubicacion proporcionada por el usuario
    for doc in coleccion.find({'id': identificador}):
        coord = doc
        coordenadas = coord[loc]['coordinates']

        center_lat = coordenadas[1]
        center_lon = coordenadas[0]

#Realizamos la consulta de la estacion más cercana
consulta = {"start station coordinate": {"$near": {"$geometry":
{"type":"Point", "coordinates":coordenadas }, "$maxDistance": 2000}}}

# Notemos que la consulta la hacemos sobre la BD de las bicis
for doc in collection.find(consulta,{'station name':1}).limit(1):
    print("El nombre de la estación más cercana a la ubicación
        + "que proporcionó es:",
            doc['station name'])
stacionopk = doc['station name']
# Buscamos las k estaciones cercanas de los
# lugares que el individuo así designe
estacionk = stations.loc[stations.loc[:, 'station name'] == stacionopk]
misEscalas_cerca.append(estacionk)
resultOp1 = pd.concat(misEscalas_cerca)
df= resultOp1

df
# le mostramos así la ruta de las estaciones cercanas
df = df.append(df.iloc[0], ignore_index=True)
#agregamos el primer valor del dataframe para hacer el circuito
print(df)

index_users = usuarios
#index_users.iloc[1]['house latitude']

#inicializamos el mapa
locationmap = folium.Map(location=[ center_lat , center_lon ],
                        zoom_start=14, width=1800, height=600 )

longitud = len(df )

```

```

#introducimos las coordenadas de nuestro usuario que
# saldra en flechita azul
folium.Marker(location=[center_lat,center_lon],
               popup='Bienvenido <b>Ud. esta aqui</b>').add_to(locationmap)

# trazamos toda la ruta
latitudes = []
for i in range ( longitud ):
    lng = float(df.iloc[i]['longitude'])
    lat = float(df.iloc[i]['latitude'])
    coord = (lat,lng)
    latitudes.append(coord)
    description = 'Stations: ' + str(df.iloc[i]['station name']) +
    '<br> Empty slots: ' + str(df.iloc[i]['tripduration'])
    folium.Marker( [ lat , lng ],
                  popup= description,
                  icon=folium.Icon(color='red')).add_to(locationmap)
    folium.PolyLine(latitudes).add_to(locationmap)
#Finalmente imprimimos en el mapa la ruta
print("La ruta ")
display(locationmap)

```

```

elif opMenu == '6':
    print('-'*5," Recomendaciones de rutas ", '-'*5, '\n')

    # el usuario debe de introducir el tiempo minimo en que
    # quiere que dure su viaje
    tRedondo = int(input("Introduzca el tiempo mínimo que le gustaría que
                        + "dure el traslado de una estación a otra: "))
    filters = {'tripduration' : {'$gte' : tRedondo }}
    # $gte = "greater than or equal"
    fields = { '_id', 'latitude', 'longitude', 'station name', 'tripduration',
               'latitude_f', 'longitude_f', 'station name_f' }

    #realizamos un query sobre colecccion maps donde queremos los
    #viajes que duren más que tRedondo
    query = list( coleccionMaps.find( filters , fields ) )
    df = pd.DataFrame ( query )

    #introducimos cuantas escalas habra en el viaje
    escalas = int(input("Introduzca el número de escalas
                        + "que deseas realizar: "))
    df = df.head(escalas)
    df = df.append(df.iloc[0], ignore_index=True)
    #agregamos el primer valor
    # del dataframe para hacer el circuito
    print('+'*5," Tu ruta es la siguiente", '+'*5, '\n')
    print(df[['longitude', 'latitude', 'station name', 'tripduration']])

    index_users = usuarios
    #index_users.iloc[1]['house latitude']

    user_ub = int(input("Introduzca el ID del usuario: "))
    ubicacion = str(input("Introduzca el nombre de su ubicación: "))
    loc = ubicacion + " coordinate"

```

```

# Obtenemos las coordenadas de la ubicacion
# proporcionada por el usuario
for doc in coleccion.find({'id': user_ub}):
    coord = doc
coordenadas = coord[loc]['coordinates']

center_lat = coordenadas[1]
center_lon = coordenadas[0]

#inicializamos el mapa
locationmap = folium.Map(location=[ center_lat , center_lon ],
                           zoom_start=14, width=1800, height=600 )

longitud = len( df )

#introducimos las coordenadas de nuestro usuario
# que saldra en flechita azul
folium.Marker(location=[center_lat,center_lon],
               popup='Bienvenido <b>Ud. esta aqui</b>').add_to(locationmap)

# trazamos toda la ruta
latitudes = []
for i in range ( longitud ):
    lng = float(df.iloc[i]['longitud'])
    lat = float(df.iloc[i]['latitude'])
    coord = (lat,lng)
    latitudes.append(coord)
    description = 'Stations: ' + str(df.iloc[i]['station name'])
    + '<br> Empty slots: ' + str(df.iloc[i]['tripduration'])
    folium.Marker( [ lat , lng ],
                   popup= description,
                   icon=folium.Icon(color='red')).add_to(locationmap)
    folium.PolyLine(latitudes).add_to(locationmap)

print("Una ruta recomienda que dura el tiempo mínimo dado de
      + "traslado de una estación a otra >>")
# Revelamos el mapa
display(locationmap)

elif opMenu == '7':
    print('-'*5," Crea tu propia ruta ", '-'*5, '\n')
    estacion_inicial = input("Introduzca el nombre de la estación
                             + "de bicicleta de la que partirá: ")

    misEscalas = []
    estacion1 = stations.loc[stations.loc[:, 'station name'] == estacion_inicial]
    misEscalas.append(estacion1)
    estacion_final = input("Introduzca el nombre de la estación de
                             + "bicicleta a la que quiere llegar: ")
    nEscalas = int(input("introduzca el número de escalas a realizar >> "))
    for i in range(nEscalas):
        nombre = input("Introduzca el nombre de la estación {0} a visitar:
                        ".format(i+1))
        estacion2 = stations.loc[stations.loc[:, 'station name'] == nombre]
        misEscalas.append(estacion2)
    estacion3 = stations.loc[stations.loc[:, 'station name'] == estacion_final]
    misEscalas.append(estacion3)
    # le proporcionamos al usuario la ruta con las estaciones que introdujo
    resultados = pd.concat(misEscalas)
    df = resultados #chechamos el df

df

```

```

df = df.append(df.iloc[0], ignore_index=True)
#agregamos el primer valor del dataframe
# para hacer el circuito

index_users = usuarios
#index_users.iloc[1]['house latitude']
user_ub = int(input("Introduzca el ID del usuario: "))
ubicacion = str(input("Introduzca el nombre de su ubicación: "))
loc = ubicacion + " coordinate"
# Obtenemos las coordenadas de la ubicacion proporcionada por el usuario
for doc in coleccion.find({'id': user_ub}):
    coord = doc
    coordenadas = coord[loc]['coordinates']

center_lat = coordenadas[1]
center_lon = coordenadas[0]

#inicializamos el mapa
locationmap = folium.Map(location=[ center_lat , center_lon ],
                          zoom_start=14, width=1800, height=600 )
longitud = len( df )

#introducimos las coordenadas de nuestro usuario
# saldra en flechita azul
folium.Marker(location=[center_lat,center_lon],
              popup='Bienvenido <b>Ud. esta aqui</b>').add_to(locationmap)

# trazamos toda la ruta
latitudes = []
for i in range (longitud-1):
    lng = float(df.iloc[i]['longitude'])
    lat = float(df.iloc[i]['latitude'])
    coord = (lat,lng)
    latitudes.append(coord)
    description = 'Stations: ' + str(df.iloc[i]['station name'])
    + '<br> Empty slots: ' + str(df.iloc[i]['tripduration'])
    folium.Marker( [ lat , lng ],
                  popup= description,
                  icon=folium.Icon(color='red')).add_to(locationmap)
    folium.PolyLine(latitudes).add_to(locationmap)

print("Esta es la ruta que se ha generado con las
      +\"estaciones que has introducido >>")
display(locationmap)

elif opMenu == '8':
    break;

else:
    print("Opción no válida. Por favor, vuelva a intentarlo.")
    input("\n\n Presione ENTER para continuar")
    print()

```

# Funcionamiento de la aplicación

A continuación mostramos algunas imagenes de como es que funciona de manera general de la aplicación con imagenes que obtuvimos de correr nuestra aplicación.

## Opción 1

El usuario es capaz de introducir un nuevo usuario dentro de la base de datos introduciendo sus datos personales

```
+++++++ MENÚ ++++++

1.- Ingresar nuevo usuario
2.- Ingresar una nueva ubicación a un usuario
3.- Localizar módulo(s) más cercano(s)
4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
5.- Viaje redondo
6.- Recomendaciones de rutas
7.- Crea tu propia ruta
8.- Salir

Ingrese una opción >> 1
----- Ingresar datos de usuario -----

Introduzca el nombre completo del usuario nuevo: Alejandro Pimentel
Coordenadas de su casa
Introduzca la latitud de su casa: 40.750113
Introduzca la longitud de su casa: -73.988085

Presione ENTER para continuar
```

## Opción 2

El usuario es capaz de introducir nuevos puntos de donde le gustaría realizar consultas y visualizar posibles rutas, en este caso introducimos las siguientes ubicaciones: `gym`, `school`, `library`, `club`

```
+++++++ MENÚ ++++++

1.- Ingresar nuevo usuario
2.- Ingresar una nueva ubicación a un usuario
3.- Localizar módulo(s) más cercano(s)
4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
5.- Viaje redondo
6.- Recomendaciones de rutas
7.- Crea tu propia ruta
8.- Salir

Ingrese una opción >> 2
----- Ingresar una nueva ubicación a un usuario -----

Introduzca el id del usuario: 1050
Coordenadas de la nueva ubicación
Introduzca el nombre de la nueva ubicación: gym
Introduzca la latitud de la nueva ubicación: 40.715555
Introduzca la longitud de la nueva ubicación: -73.954276

Presione ENTER para continuar
```

+++++ MENÚ +++++

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingrese una opción >> 2

----- Ingresar una nueva ubicación a un usuario -----

Introduzca el id del usuario: 1050

Coordenadas de la nueva ubicación

Introduzca el nombre de la nueva ubicación: school

Introduzca la latitud de la nueva ubicación: 40.714392

Introduzca la longitud de la nueva ubicación: -73.960111

Presione ENTER para continuar

+++++ MENÚ +++++

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingrese una opción >> 2

----- Ingresar una nueva ubicación a un usuario -----

Introduzca el id del usuario: 1050

Coordenadas de la nueva ubicación

Introduzca el nombre de la nueva ubicación: library

Introduzca la latitud de la nueva ubicación: 40.720044

Introduzca la longitud de la nueva ubicación: -73.960358

Presione ENTER para continuar

```

+++++ MENÚ +++++
1.- Ingresar nuevo usuario
2.- Ingresar una nueva ubicación a un usuario
3.- Localizar módulo(s) más cercano(s)
4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
5.- Viaje redondo
6.- Recomendaciones de rutas
7.- Crea tu propia ruta
8.- Salir

Ingrese una opción >> 2
----- Ingresar una nueva ubicación a un usuario -----

Introduzca el id del usuario: 1050
Coordenadas de la nueva ubicación
Introduzca el nombre de la nueva ubicación: club
Introduzca la latitud de la nueva ubicación: 40.705400
Introduzca la longitud de la nueva ubicación: -73.960909

Presione ENTER para continuar

```

## Opción 3

De las ubicaciones que le usuario definio en la opción 2, este es capaz de identificar las estaciones de bicicleta más cercanas.

```

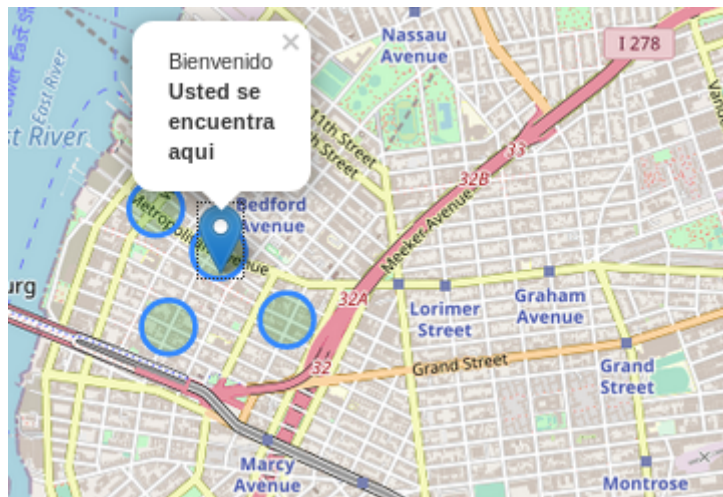
+++++ MENÚ +++++
1.- Ingresar nuevo usuario
2.- Ingresar una nueva ubicación a un usuario
3.- Localizar módulo(s) más cercano(s)
4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
5.- Viaje redondo
6.- Recomendaciones de rutas
7.- Crea tu propia ruta
8.- Salir

Ingrese una opción >> 3
----- Localizar módulo más cercano -----

Introduzca el ID del usuario: 1050
Introduzca el nombre de la ubicación: school
Introduzca la cantidad de estaciones de bicicleta que desea obtener: 4
El nombre de la(s) estacion(es) más cercana(s) (de menor a mayor distancia)
a la ubicación que proporcionó es(son):
-Metropolitan Ave & Bedford Ave
-S 3 St & Bedford Ave
-Grand St & Havemeyer St
-Wythe Ave & Metropolitan Ave
station name
0      Grand St & Havemeyer St
1      S 3 St & Bedford Ave
2      Metropolitan Ave & Bedford Ave
3      Wythe Ave & Metropolitan Ave

```





## Opción 4

De acuerdo a un tiempo en el que quiere viajar, el usuario es capaz de visualizar las rutas cercanas que cumplen con el plazo de tiempo que especificó.

+++++ MENÚ +++++

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingrese una opción >> 4

----- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar -----

- 1.- Planear viajes usando como posible salida las estaciones más cercanas de la ubicación del usuario
- 2.- Planear viajes usando como posible salida las estaciones que usted nos proporcione

Introduzca una opción: 1

Introduzca el ID del usuario: 330

Introduzca el nombre de la ubicación: gym

Introduzca la cantidad de estaciones de bicicleta que desearía obtener: 4

El nombre de la(s) estación(es) más cercana(s) (de menor a mayor distancia)

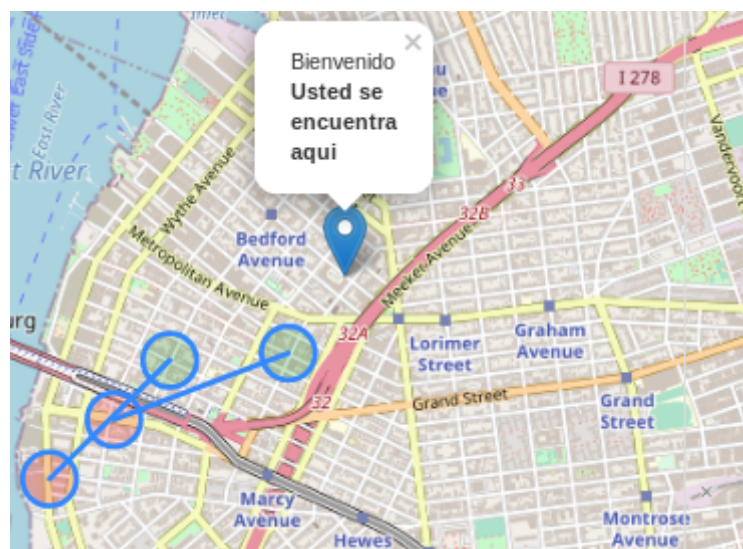
a la ubicación que proporcionó es(son):

- Grand St & Havemeyer St
- Metropolitan Ave & Bedford Ave
- Wythe Ave & Metropolitan Ave
- S 3 St & Bedford Ave

Introduzca el tiempo aproximado (en segundos) que quiere que dure su viaje: 330

De las estaciones obtenidas como inicio del viaje, las que tienen una ruta con un tiempo aproximado al proporcionado son:

	station name	tripduration	station name_f
0	Grand St & Havemeyer St	361	Broadway & Berry St
1	S 3 St & Bedford Ave	360	Kent Ave & S 11 St





# \*\*\*\*\* MENÚ \*\*\*\*\*

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingresa una opción >> 4

----- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar -----

- 1.- Planear viajes usando como posible salida las estaciones más cercanas de la ubicación del usuario
- 2.- Planear viajes usando como posible salida las estaciones que usted nos proporcione

Introduzca una opción: 2

Introduzca la cantidad de estaciones de bicicleta que desearía obtener como posible ruta: 5

Introduzca el nombre de las estaciones:

Introduzca el nombre de la estación 1: Division St & Bowery

Introduzca el nombre de la estación 2: Hancock St & Bedford Ave

Introduzca el nombre de la estación 3: Broadway & W 24 St

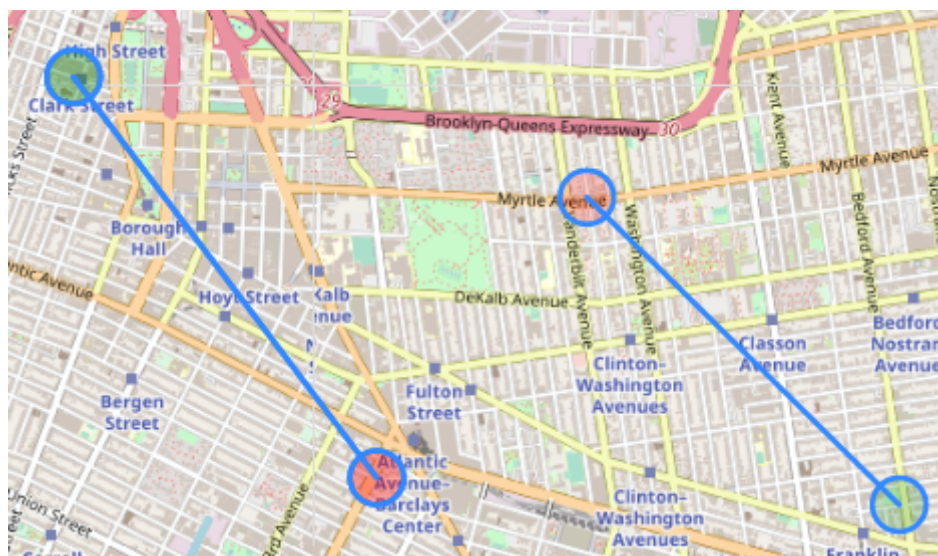
Introduzca el nombre de la estación 4: E 23 St & 1 Ave

Introduzca el nombre de la estación 5: Clark St & Henry St

Introduzca el tiempo aproximado (en segundos) que quiere que dure su viaje: 715

De las estaciones obtenidas como inicio del viaje, las que tienen una ruta con un tiempo aproximado al proporcionado son:

	station name	tripduration	station name_f
0	Clark St & Henry St	659	Dean St & 4 Ave
1	Hancock St & Bedford Ave	774	Clinton Ave & Myrtle Ave
2	Broadway & W 24 St	752	St Marks Pl & 2 Ave
3	E 23 St & 1 Ave	730	E 20 St & Park Ave



# Opción 5

El usuario es capaz de generar un viaje redondo utilizando las estaciones más cercanas a las ubicaciones que introdujo en la opción 2, al ser un viaje redondo se le regresa al usuario un circuito completo entre las ubicaciones que desee visitar.

```
+++++ MENÚ +++++
1.- Ingresar nuevo usuario
2.- Ingresar una nueva ubicación a un usuario
3.- Localizar módulo(s) más cercano(s)
4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
5.- Viaje redondo
6.- Recomendaciones de rutas
7.- Crea tu propia ruta
8.- Salir

Ingrese una opción >> 5
----- Viaje Redondo -----

Introduzca el ID del usuario: 1050
Introduzca el nombre de la ubicación: school
El nombre de la estación más cercana a la ubicación que proporcionó es: Metropolitan Ave & Bedford Ave
¿A cuántos de sus lugares registrados desea ir? 3
Introduzca el nombre de la ubicación 1: gym
El nombre de la estación más cercana a la ubicación que proporcionó es: Grand St & Havemeyer St
Introduzca el nombre de la ubicación 2: library
El nombre de la estación más cercana a la ubicación que proporcionó es: Wythe Ave & Metropolitan Ave
Introduzca el nombre de la ubicación 3: club
El nombre de la estación más cercana a la ubicación que proporcionó es: Bedford Ave & S 9th St
  latitude longitude ... outgoing trips all trips
0  40.715348 -73.960241 ...           37       72
1  40.712868 -73.956981 ...           8        17
2  40.716887 -73.963198 ...          15        32
3  40.708531 -73.964090 ...           2         2
4  40.715348 -73.960241 ...          37        72

[5 rows x 10 columns]
La ruta
```



# Opción 6

Con respecto a un tiempo que el usuario desea viajar, se le devuelve una recomendación de una posible ruta a lo largo de la ciudad.

+++++ MENÚ +++++

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingrese una opción >> 6

----- Recomendaciones de rutas -----

Introduzca el tiempo mínimo que le gustaría que dure el traslado de una estación a otra: 600

Introduzca el número de escalas que desee realizar: 4

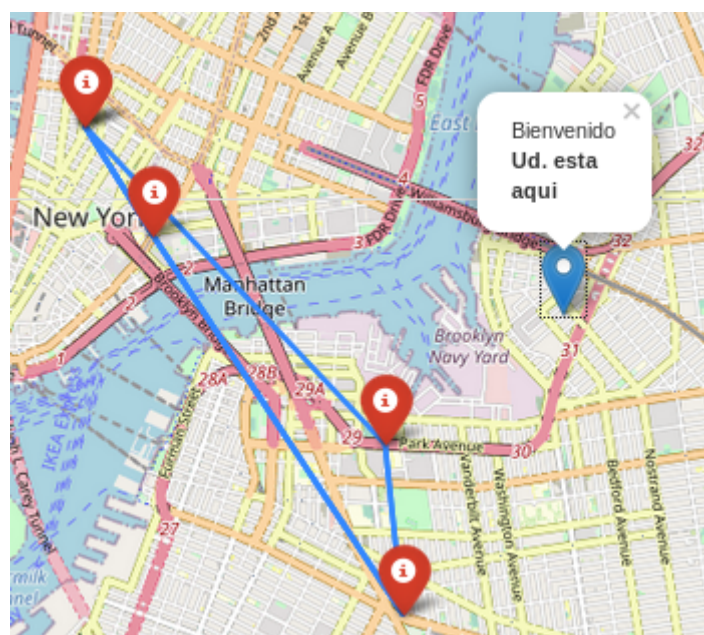
+++++ Tu ruta es la siguiente +++++

	longitude	latitude	station name	tripduration
0	-74.006667	40.719116	Franklin St & W Broadway	956
1	-74.000165	40.711174	St James Pl & Pearl St	1307
2	-73.976323	40.683826	Atlantic Ave & Fort Greene Pl	660
3	-73.978034	40.696089	Park Ave & St Edwards St	806
4	-74.006667	40.719116	Franklin St & W Broadway	956

Introduzca el ID del usuario: 1050

Introduzca el nombre de su ubicación: club

Una ruta recomienda que dura el tiempo mínimo dado de traslado de una estación a otra >>



## Opción 7

El usuario puede crear su propia ruta entre dos puntos diferentes definiendo estación por estación su propia ruta (en caso de que esta no exista en la base de datos)



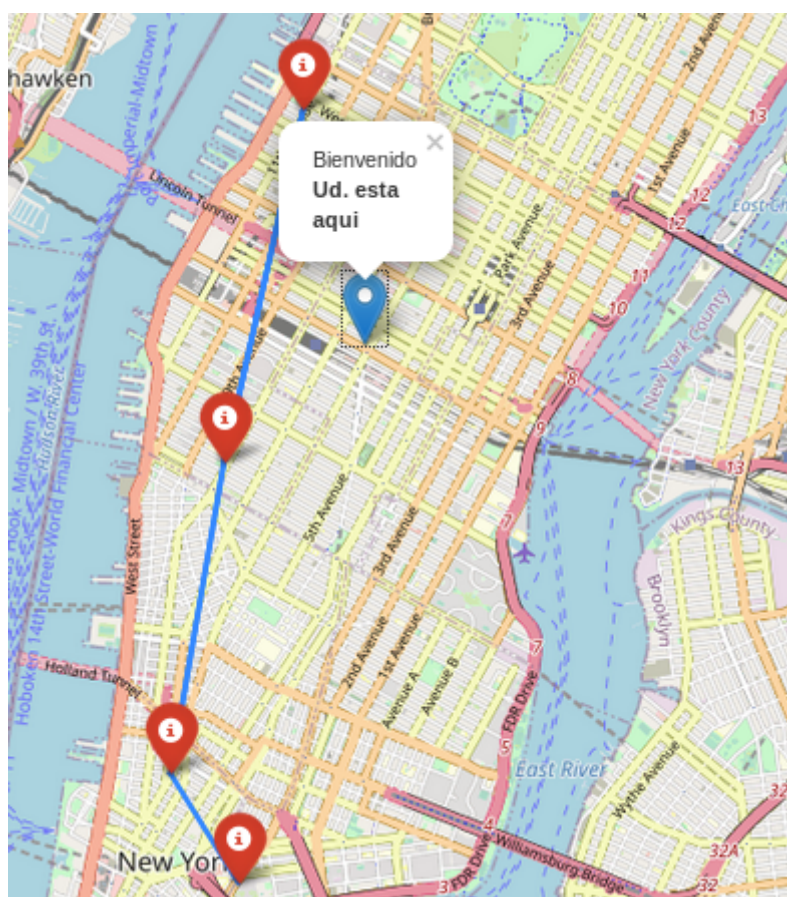
+++++ MENÚ +++++

- 1.- Ingresar nuevo usuario
- 2.- Ingresar una nueva ubicación a un usuario
- 3.- Localizar módulo(s) más cercano(s)
- 4.- Planear viajes, indicando el tiempo aproximado en el que se quiere viajar
- 5.- Viaje redondo
- 6.- Recomendaciones de rutas
- 7.- Crea tu propia ruta
- 8.- Salir

Ingrese una opción >> 7

----- Crea tu propia ruta -----

Introduzca el nombre de la estación de bicicleta de la que partirá: W 52 St & 11 Ave  
Introduzca el nombre de la estación de bicicleta a la que quiere llegar: St James Pl & Pearl St  
introduzca el número de escalas a realizar >> 2  
Introduzca el nombre de la estación 1 a visitar: W 17 St & 8 Ave  
Introduzca el nombre de la estación 2 a visitar: Franklin St & W Broadway  
Introduzca el ID del usuario: 2001  
Introduzca el nombre de su ubicación: house  
Esta es la ruta que se ha generado con las estaciones que has introducido >>



## CONCLUSIONES

Como pudimos observar, no todos los documentos deben tener los mismos campos. Aunque en una nuestra colección que contenía a los usuarios inicialmente todos tenían la ubicación; algunos podrían tener la ubicación de otro lugar distinto. Esto es, si lo relacionamos a tabla, algunos documentos (registros) podrían tener más columnas que otros sin afectar a los demás. Es aquí donde se presenta la versatilidad de trabajar con MongoDB. Además, aunque en MongoDB no hay JOINS, pudimos relacionar dos colecciones; poniendo el resultado de una consulta sobre una colección en otra consulta que le haríamos a una colección distinta (Cuando buscábamos las estaciones más cercanas, le pedíamos el nombre de la ubicación al usuario. Una vez que obteníamos sus coordenadas, la búsqueda era sobre la colección que contenía la información de las estaciones de las bicicletas, y no sobre la colección que tenía la información de los usuarios).

Además, una de las cosas que más nos agradó, fue cuando buscábamos estaciones cercanas a lugares específicos, había una sintaxis en especial que hacerlo, sin necesidad de que nosotros implementááramos alguna medida de distancia (como Manhattan, euclidiana, etc.)

Sin embargo, podemos ser más generales con las ventajas más importantes de trabajar con MongoDB:

## Ventajas

### 1.- Compartir y balancear la carga:

Cuando tenemos que lidiar con grandes volúmenes de datos para administrar la distribución del tráfico a varias máquinas para hacer el equilibrio de carga, entonces existe una ventaja significativa de usar MongoDB sobre las bases de datos convencionales. Sharding es un enfoque único de MongoDB, que ayuda a cumplir con las demandas relacionadas con el crecimiento de datos y permite el escalado horizontal en múltiples máquinas para acomodar el creciente volumen de datos.

### 2.- Alta flexibilidad:

Usar MongoDB es más simple y más versátil en comparación con otros RDBMS, ya que no necesita una estructura de datos unificada en todos los objetos. Sin embargo, para los usuarios, la estructura de datos y la consistencia son algo bueno, por lo que algunas veces es recomendable tener una estructura de datos unificada.

### 3.- Velocidad

Como los datos se almacenan en un solo punto, MongoDB funciona muy rápido. Pero, por otro lado, funciona de esta manera solo si todos los datos que maneja son un solo documento. Mientras se trabaja en un dato que es un modelo relacional, puede llevar más tiempo manejar consultas independientes para recuperar datos.

### 4.- Podemos elegir qué nivel de consistencia deseamos dependiendo del valor de los datos:

Por ejemplo, un rendimiento más rápido: dispere y olvide las inserciones en MongoDB, un rendimiento más lento: espere hasta que la inserción se haya replicado en varios nodos antes de volver.

Cabe destacar que no todo es gratis, pues a veces tenemos que sacrificar algunas cosas. Por lo tanto, también podemos mencionar algunas de sus ventajas más destacadas:

## Desventajas

### 1.- Más uso de memoria:

Al almacenar un nombre clave para cada documento, MongoDB necesita un mayor espacio de memoria.

### 2.- No hay JOIN:

A diferencia de las Bases de Datos relacionales, las uniones no son posibles con MongoDB. Debemos realizar múltiples consultas si es necesario unir, lo que debe hacerse manualmente.

### 3.- Aún se encuentra en desarrollo:

MongoDB es una plataforma relativamente nueva que todavía está en desarrollo, mientras que SQL estaba bien establecido en la década de 1980. MongoDB tampoco está ampliamente documentado y también enfrenta dificultades con la falta de soporte.

### 4.- Sin soporte para transacciones:

se admiten ciertas operaciones atómicas, en un solo nivel de documento

Teniendo en cuenta todos los factores anteriores, podemos ver que hay muchos beneficios y desventajas en la adopción de MongoDB como nuestra DBMS. Debemos pensar seriamente en activarlo solo si las ventajas pueden beneficiarnos significativamente y las desventajas pueden no afectarlo mucho. Además, como hemos visto, las bases de datos relacionales tradicionales están allí en la práctica durante mucho tiempo, y MongoDB es relativamente nuevo en la industria de administración de bases de datos. Sin duda, tiene un gran potencial, pero es posible que tengamos que esperar unos años más para verlo completamente maduro y funcional para enfrentar de manera efectiva los desafíos de gestión de la base de datos. Existe una mayor posibilidad de que MongoDB se convierta en una plataforma ventajosa en la industria de bases de datos en el futuro.

In [ ]: